The role of metaphor in user interface design

Thesis

The Role of Metaphor in User Interface Design

Mark Treglown

MSc Applied Artificial Intelligence, University of Aberdeen, 1991

BSc (Hons.) Computer Science, University of Sussex, 1990

Submitted for the degree of Doctor of Philosophy in Human-Computer Interaction

Institute of Educational Technology, The Open University, Milton Keynes.

December 1999

EX12

THE OPEN UNIVERSITY

RESEARCH SCHOOL

Library Authorisation Form

Please return this form to the Research School with the two bound copies of your thesis to be deposited with the University Library. All candidates should complete parts one and two of the form. Part three only applies to PhD candidates.

Part One: Candidates Details

Name: MARK TREGLOWN PI: M7110726

Degree: PhD

Thesis title: THE ROLE OF METAPHOR IN USER INTERFACE DESIGN

Part Two: Open University Library Authorisation

I confirm that I am willing for my thesis to be made available to readers by the Open University Library, and that it may be photocopied, subject to the discretion of the Librarian.

Signed: Mark Treglown Date: 4 May 2003

Part Three: British Library Authorisation [PhD candidates only]

If you want a copy of your PhD thesis to be available on loan to the British Library Thesis Service as and when it is requested, you must sign a British Library Doctoral Thesis Agreement Form. Please return it to the Research School with this form. The British Library will publicise the details of your thesis and may request a copy on loan from the University Library. Information on the presentation of the thesis is given in the Agreement Form.

Please note the British Library have requested that theses should be printed on one side only to enable them to produce a clear microfilm. The Open University Library sends the fully bound copy of theses to the British Library.

The University has agreed that your participation in the British Library Thesis Service should be voluntary. Please tick either (a) or (b) to indicate your intentions.

☑ I am willing for the Open University to loan the British Library a copy of my thesis.
A signed Agreement Form is attached

☐ I do not wish the Open University to loan the British Library a copy of my thesis.

Signed: Mark Treglown Date: 4 May 2003

Abstract

The thesis discusses the question of how unfamiliar computing systems, particularly those with graphical user interfaces, are learned and used. In particular, the approach of basing the design and behaviour of on-screen objects in the system's model world on a coherent theme and employing a metaphor is explored. The drawbacks, as well as the advantages, of this approach are reviewed and presented. The use of metaphors is also contrasted with other forms of users' mental models of interactive systems, and the need to provide a system image from which useful mental models can be developed is presented.

Metaphors are placed in the context of users' understanding of interactive systems and novel application is made of the Qualitative Process Theory (QPT) qualitative reasoning model to reason about the behaviour of on-screen objects, the underlying system functionality, and the relationship between the two. This analysis supports re-evaluation of the domains between which user interface metaphors are said to form mappings. A novel user interface design, entitled Medusa, that adopts guidelines for the design of metaphor-based systems, and for helping the user develop successful mental models, based on the QPT analysis and an empirical study of a popular metaphor-based system, is described. The first Medusa design is critiqued using well-founded usability inspection method.

Employing the Lakoff/Johnson theory, a revised version of the Medusa user interface is described that derives its application semantics and dialogue structures from the entailments of the knowledge structures that ground understanding of the

interface metaphor and that capture notions of embodiment in interaction with computing devices that QPT descriptions cannot. Design guidelines from influential existing work, and new methods of reasoning about metaphor-based designs, are presented with a number of novel graphical user interface designs intended to overcome the failings of existing systems and design approaches.

Acknowledgements

My grateful thanks go to my supervisor Tim O'Shea, for his advice, encouragement, editorial advice, and for his patience.

I would like to thank Debbie Stone and Vanessa Evers for proofreading previous drafts, and for their invaluable comments. The work reported in this thesis benefited from conversations with Gregory Abowd, Mark Elsom-Cook, Alan Dix, Thomas Green, Kim Issroff, Tim O'Shea, Steve Schneider, Randall Smith, and Debbie Stone among many others. Thank you to Matt Smith, Don Clark and Mark Elsom-Cook for lending me hardware and office space to conduct the MacLearning study reported in this thesis. Many thanks and sincere apologies are due to Richard Bornat, Gill Ritchie, Pete Woodward, Ben du Boulay, Des Watson and Roger Sinnhuber for help with empirical work that eventually proved impossible to conduct.

I would like to thank Simon Holland who was responsible in large part for my studying at the Open University. Thank you to Eileen Scanlon and all in the CALRG. Thanks are due to Marian Petre for providing Sun workstations, and to Yibing Li for making sure they ran more often than not. I would like to thank Dave Perry, Jon Oliver, Dave Signorini, Rob Griffiths and Colin Pink for technical support. Thank you to everyone in cyberspace who provided advice, references, code fragments and who sent useful papers; keep on rockin' in the free world. Louis Feinberg and Jerome and Maurice Horowitz also provided invaluable assistance.

"Technical texts are generally understood to report work that their authors have done; they are focused on machinery in a broad sense, be it hardware, software, or mathematics. They open by making claims - 'Our machinery can do such and such and others' cannot' - and they confine themselves to demonstrating these claims in a way that others can replicate. They close by sketching further work - more problems, more solutions. Critical texts, by contrast, *are* the work that their authors have done. Their textuality is in the foreground, and they are focused on theoretical categories. They open by situating a problematic in an intellectual tradition, and they proceed by narrating their materials in a way that exhibits the adequacy of certain categories and the inadequacy of others. They close with a statement of moral purpose."

— P. E. Agre (1997) *Computation and Human Experience*, Cambridge University Press: xiii.

"This is the time, and this is the record of the time."

— Laurie Anderson (1979) *United States Live I-IV*, Warner Brothers Records.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

"So then I got down to the writing, and it was awful. I don't know why I'd ever romanticised it. I don't know why anyone would want to do it. It stinks. It's like a disease. It's an illness, writing. It steals your body from you. There's no audience. You're alone."

— Spalding Gray, from the 'Monster in a Box' monologue.

1.1 The Problem

Card, Moran, and Newell (1983: vii) claim that:

> "Designing interactive computer systems to be efficient and easy to use is important so that people in our society may realise the potential benefits of computer-based tools."

The vast majority of user interface designers and researchers of human-computer interaction (HCI) will agree with this view. What is not agreed upon, however, is how interactive systems should be designed; or how efficiency and ease of use may be designed for and how a completed system can be judged to possess them; or how one recognises a member of a society who can be expected to understand and make use of a computing system; or how systems should be designed so that they are comprehensible and usable in a particular culture or society; or indeed whether

computing systems, in fact, deliver any benefits to those who use them (Landauer, 1995). For Herbert Simon (1981), design, like all activity, is a matter of making choices from options and actions in a problem space. Confronted by numerous options, people engage in a process termed *satisficing* — making choices that are satisfactory, not necessarily those that are optimal — if they are to not be stuck in a state of paralysis, unable to decide between a number of equally valid choices. In some sense, we are more fortunate if constrained by time and the limited availability of resources. For the user interface designer, there exist many tens of design lifecycle models, interaction styles, input and output devices, programming languages, user interface toolkits, and usability evaluation techniques which can be combined in many ways during a design task. This multiplicity of choice arises because of what is claimed to be a *theory gap* in HCI (Landauer, 1989; Long and Dowell, 1989). There is no theory of user interface design that allows us to construct the best interface given a set of requirements and constraints, neither are there sufficient bodies of knowledge or of experimental data provided by cognitive psychology which can be employed to constrain design choices. If we are to discuss user interface design, we are required to constrain the types of system we examine.

The interfaces we consider are those where, in contrast to command-based interfaces where the user converses with an unseen agent in a natural or artificial language about an unseen but assumed task domain (the *conversation* paradigm of interaction), the task domain is depicted on-screen and its state may be directly altered. These systems are said to be based on the *model world* paradigm (Hutchins, Hollan, and Norman, 1986). Such systems are a subset of those systems termed *direct manipulation* (Shneiderman, 1982, 1983). Direct manipulation systems are characterised by:

- Continuous representation of the object of interest;
- Physical actions or labelled button presses instead of complex syntax;

- Rapid incremental reversible operations whose impact on the object of interest is immediately visible.

1.2 A Solution - Metaphor Recommended

Even when the types of systems we consider in this research are restricted to those that support the model world paradigm of interaction, many design methodologies exist that can be considered and employed in a design task. The approach we consider in this research is one that is recommended by many influential and best-selling texts (Hix and Hartson, 1993; Neilsen, 1993; Thimbleby, 1990), which is to use a metaphor or analogy in the depiction and programming of the behaviour of on-screen objects. Analogy is recommended as a means of understanding new concepts and problem solving in many domains, it is, for example famously advocated by Pólya (1945) as a method for mathematical problem solving. The motivation underlying user interface metaphors is that users can make use of their existing knowledge structures with little modification, making the unfamiliar interactive system easier to use and learn than if users need to be acquire new knowledge structures (Carroll and Thomas, 1982). An example of how the use of user interface metaphors is recommended to students of computer science students is given by Evanson and Holland (1996):

> "To make users feel comfortable, successful software surrounds them with pictures or icons of familiar objects. Because such environments are meant to resemble the everyday world, designers say they're using a metaphor.
>
> Good software uses metaphor, which allows people to draw on their mental models of how the world works. All screen objects should fit the metaphor and act in sympathy with the user's expectations."

1.3 A Solution? Metaphor Also Considered Harmful

While employing metaphors and analogies in the design of on-screen model worlds is often recommended, the metaphor-based user interface design process has not been described in detail by researchers. Anderson, Smyth, Knott, Bergan, Bergan, and Alty (1994) are exceptions and do provide some details as to how metaphors could be employed and how the best metaphor could be chosen from a set of alternatives. While metaphor-based design is ill-defined, there is some question as to whether metaphors do, in fact, offer the best solution to providing users with systems that are easily learned, used, and understood. As detailed in Chapter 4, previous uses of metaphor in user interfaces show that the metaphors employed give rise to serious usability problems while solving others. Criticisms of the use of metaphor in user interface design are long standing. Halasz and Moran (1982) describe the problem most often encountered with metaphors, that they *break down*. There often, if not always, exist aspects of the analogical source domain that will not carry over into the target domain, or some functionality supported by the target domain of the computing system cannot be accounted for by the user interface metaphor. In all accounts of the cognitive mechanisms underlying metaphor understanding that have been proposed as valuable in user interface design, and in all existing approaches to employing metaphors, we see that metaphors are subject to these sorts of failures and breakdowns. That metaphors are implemented on computer hardware presents additional difficulties, the behaviour of model worlds, system image or system illusion, is dictated to an unpredictable degree by the behaviour of the operating system and by the hardware on which it, and the user interface process, executes.

In addition to pragmatic difficulties, and the possibly inherent problem of breakdowns and limitations of scope in metaphors, there exist other difficulties with the use of metaphor in user interface design. The most serious arise from philosophy

and formal semantics, recent work in which fields (for example Putnam, 1981; Lakoff, 1987), in addition to a shift from long standing views of mind and cognitive science, points to a view that suggests that metaphor plays no role in understanding. This work suggests that user interfaces cannot be understood through metaphor, as metaphor is currently widely understood in user interface design.

1.4 A Solution - New Metaphors and New Approaches to Metaphor

Current trends in user interface design show a shift away from the metaphors currently widely used in desktop computing systems, toward immersive environments and desktop virtual realities, augmented realities, and visual formalisms. Also of growing importance are *spatial* metaphors, where the location of objects in the model world is more important for understanding and recognition than classification, action, and existing knowledge structures of a real world domain. Seeking to avoid the explicit use of metaphors in model worlds that are intended to account for much of the target system ignores the major part metaphor plays in understanding the real world, and by extension, in understanding model worlds.

As with many aspects of cognition, metaphor has proved to be far more complex to understand than tasks that people themselves consider difficult. With user interface metaphors though a poor design can seem as difficult for users to understand and interact with as the mental mechanisms of metaphor understanding are to the researcher. Human-computer interaction has responded to the problem of metaphor and analogy in a number of ways. As mentioned above, current trends are shifting away from designs in which the problem of addressing metaphor must be faced. As with consciousness, problems can be divided into those that are *easy* and those that

are *hard*[1]. The easy problems of explaining cognitive functions are easy because they only require the specification of a mechanism that can perform the function. The hard problem is that even after all functionality has been explained, the further question of why functions are accompanied by experience may remain. Much of the previous work on analogy and metaphor surveyed below addresses the easy problems. While some avoid either type of problem and regard metaphor in HCI as an area in which all problems have been solved or are unworthy of consideration, if user interface design is to understand the *user experience*, the hard problem of *experience in general* will eventually have to be addressed.

The solutions adopted in this research are, firstly, to develop new metaphors to the functionality and services provided by systems which existing metaphors seek to explain. Secondly, other new metaphors are based on methods of thinking about the analysis of systems and metaphors that have previously not been applied to *user interface* metaphors and human-computer interaction, or that have not previously been explored in the depth that they are in this research. These methods of thinking have a focus on human experience built in and so allow some progress on the hard problem to be made. Case studies examining existing interface design solutions, and also novel interface designs, are undertaken to illustrate the approach adopted.

1.5 Overview of the Thesis

We present the motivation for a novel user interface to facilities supported by a computer's operating system. We also present details of its design and a critique of the design based on the results of applying a usability inspection method. This

[1] This distinction is attributed to, and is frequently discussed in the writing of, the philosopher David Chalmers, for example "Facing up to Consciousness" in Rita Carter's (2002) *Consciousness*, Weidenfeld and Nicholson: 50-55.

interface, rather than employing a single real world metaphor, attempts to make mechanisms that would otherwise be implicit and would have to be inferred by users, explicit. The interface is designed with the intention that users are more able to construct a realistic mental model of the system. Important in this part of the thesis is the realisation that the difficulties presented by existing theories of metaphor understanding, and their application in design tasks, must be addressed. In addressing these difficulties, a contemporary theory of metaphor, not usually applied to user interface design, is introduced and its usefulness is explored by undertaking a number of small case studies. In these studies, aspects of novel user interfaces that prove difficult to describe and account for are examined. This theory is then employed as a predictive tool to help design a revised version of the novel user interface design presented earlier. In the revised design, the difficulties of metaphors are appreciated, but the pervasive nature of metaphor in understanding is not ignored.

Chapter Two reviews a number of existing, historically important, systems which employ metaphors and analogies in providing a user interface to the facilities offered by the operating system of a complex computing device. Those systems that had a profound impact on future commercially available systems, or on human-computer interaction research, are focused on. In particular, systems that have helped to define what is commonly understood by the use of metaphor in user interface design, or that have employed metaphors when considering interaction using novel or unfamiliar modalities, are surveyed.

Chapter Three presents the results of an empirical study of first-time users of the Apple Macintosh computer. This study was undertaken to examine the robustness of a previous similar study which explored the usability of another desktop metaphor-based user interface, and to examine the pragmatics of user interface metaphors in use in order to question the claim that interfaces based on metaphors have superior

usability. Results of the study are also used to constrain the design of the novel user interface design presented in Chapters 6 and 7.

Chapter Four examines the role of metaphors and analogies in learning to use unfamiliar computing systems. Analogy plays an important role in learning and problem solving. Users will often make use of existing skills and knowledge and may make spontaneous analogous connections when they are confronted with a new system. The use of analogies and metaphors is not without difficulties however. In this chapter the drawbacks of specific metaphors; the difficulties that arise when specific theories of metaphor are applied in an attempt to understand what role metaphor plays in HCI; and the difficulties of attempting to evaluate chosen user interface metaphors, are surveyed. This chapter surveys the theories of metaphor that have previously been, or which can be, employed to design, criticise, or reason about the usability of, user interfaces. Realising the drawbacks of existing theories of metaphor comprehension and the limitations of other forms of mental model description, recent work undertaken by George Lakoff and his colleagues, including Mark Johnson, on metaphor comprehension is also considered in this chapter. Application of Lakoff and Johnson's work as a candidate approach to user interface metaphor is then presented.

Chapter Five places metaphors in the context of other forms of mental models that users may possess and employ when interacting with computing systems. It is found that many approaches to aiding users by providing them with an account of how computing systems work also rely on metaphors. This chapter also seeks to stress the importance of users having useful knowledge of how a computing system works. In addition to knowing how their tasks should be performed, knowing how the device works is useful if interaction with a system is to be successful; if methods for performing new tasks are to be generated; if unexpected system behaviour is to be explicable and, where needed, correcting tasks must be performed. Using a

qualitative reasoning model and notation developed in the field of artificial intelligence, a novel attempt to model the behaviour of objects in model worlds, and to analyse the relationship between objects in the model world and their implementation as functionality in the underlying software, is presented. This method of analysis supports a revised view of the domains between which analogical mappings should be thought of as being made between when graphical user interfaces are required to be understood. This model also reveals and captures the mismatches between user interfaces based on a physical world metaphor and the actual behaviour exhibited by these systems based on a physical world metaphor. It also suggests, as with other means of capturing mental models, that knowledge of the underlying functionality, and its actual, and temporal, behaviour is required if user interfaces are to be understood fully. Sections of this chapter have been previously published as (Treglown, 1994). The ability of the Lakoff/Johnson theory of metaphor understanding to provide accounts of how problematic features of existing user interfaces can be understood, or shown to be inherently difficult to use, is demonstrated in a number of case studies presented in this chapter. Sections of this chapter have been previously published as (Treglown, 1999; 2000; 2001).

Chapters Six and Seven present the design of the Medusa system, the motivation for which arises from, in particular, the design guidelines discussed in Chapter Five. The system architecture and details of a proposed implementation are presented along with relevant aspects of the system's specification. Medusa provides a graphical user interface to the application programmer interface of the operating system of a complex computing device. Medusa also provides a representation of the computer's file space and supports file organisation and retrieval tasks. Medusa adopts three principles that are applied consistently to every relevant instance of the classes of on-screen objects. These principles are, firstly, the idea of self-representation in icon design, where the final form in which a data file is presented is used to generate a rich icon design for the file. User interface design is not the only discipline in which

metaphors have been commonly used to account for the concepts that make up the domain of interest. In cognitive science, particularly in naive psychology, for example, metaphors have frequently been used to explain mental states and behaviour. Rumelhart (1989: 298) claims that "... the serial processing Von Neumann computer has become the dominant approach to the understanding of higher mental processes over the past 25 years or so." Rumelhart complains that while the metaphor has a great deal of merit, and improved upon many conceptualisations of the mind that preceded it, the conceptual baggage carried by the computer metaphor for the brain has limited, or must inevitably limit, further progress in understanding. Rumelhart suggests that more brain-like metaphors must replace computer-like metaphors to account for the brain. Rumelhart (1989: 299), in short, wants to "... replace the computer metaphor with the brain metaphor". In the design of Medusa , therefore, we seek for users to have greater understanding of the system by replacing traditional user interface metaphors with a computer metaphor to explain a computing system.. This involves providing sufficient description and depiction on-screen of the device components, their interconnections, and their dynamic behaviour to permit users to easily alter accessible system parameters. In addition, unexpected system behaviour should be noticeable and explained in a way that refers to the state of the hardware and operating system, but which does not require breaking the model world's metaphor. The third principle is consistency of task sequences, the same interaction style is adopted to permit interaction with every instance and class of on-screen object.

Chapter Eight presents a critique of the Medusa system design described in Chapters Six and Seven. As no complete working prototype of the Medusa system exists, low-cost usability inspection methods are employed to examine the usability of the system design. The usability inspection method chosen is the cognitive walkthrough method. This technique is termed an inspection method rather than an evaluation method because no user studies are conducted, it is, though, a technique proven to be

able to determine the usability of a system when users attempt to perform tasks that the system is designed to support. A number of cognitive walkthroughs are reported which consider realistic interaction tasks and which demonstrate the general usability of the Medusa system. The walkthroughs, however, also reveal some minor usability difficulties and the omission of any interface features that support *recovery*, being able to reach a desired system state after performing an erroneous action. Possible design solutions to support recovery are discussed in Chapter 10. Design solutions addressing other failings of the Medusa system are presented in Chapter 9, the means of analysis used to justify these solutions are presented in Chapters 4 and 5.

Chapter Nine presents details of the design of a second, revised, version of Medusa which is based upon employing the account of metaphor understanding presented in Chapter Eight as a predictive and critical tool. Motivation for another revised version of Medusa, entitled Medusa-τ, is presented. Medusa-τ is closely related to the Medusa system, it retains much of the Medusa system's design, but additional requirements are considered. These are intended to address the breakdowns in system behaviour and understanding that occur due to the uncertain temporal behaviour of computing hardware and its user interface. Two approaches to addressing the problems of breakdown in interface behaviour can be proposed, one can give the user an explanatory account of the source of the breakdown, or one can attempt to prevent, through appropriate hardware and software technology, the breakdown from occurring. Medusa-τ employs on-going work in formal specification of user interface software, software architectures, and the choice, and possible development, of appropriate programming languages to prevent temporal breakdowns in the behaviour of on-screen objects where possible. This on-going

work is described. Sections of this chapter were accepted for publication[2] but are to date unpublished.

Chapter Ten concludes and summarises the thesis. The contributions of the work are described and suggestions for further work are presented.

[2] At the International Workshop on Physicality and Tangibility in Interaction, (Sienna, Italy, 20-22 October 1999).

Chapter 2

Existing Approaches to the Use of Metaphor and Analogy in User Interface Design

"We didn't have metaphors when I was young. We didn't beat about the bush."

— Fred Trueman.

2.1 Introduction

Learning to use an unfamiliar computer system can take a considerable time as users acquire the knowledge required to use the system successfully. Carroll and Thomas (1982) state that the relevant knowledge structures cannot, by definition, be accessed at first, instead related knowledge is accessed and forms a metaphor for the knowledge being acquired. Users are often found to devise and employ metaphors when learning a previously unfamiliar computer system (Payne, 1991a), but systems designers can often aid users by making the metaphor to suitable related knowledge explicit in the model world represented on-screen. Metaphors employed in user interfaces tend to be *copula*, directive, instructional, statements of the form "X is (like) a Y". According to these assertions, an unfamiliar domain, X, can be explained by making its similarities to a familiar domain, Y, explicit.

Metaphors employed in user interfaces may be employed to describe some small aspect of the system, or a single application. The lightbox metaphor (Lüdtke and Nackunstz, 1987), for example, has been employed to present X-ray data, and the

note card metaphor (Halasz et al., 1987) has been employed in hypertext systems. Metaphors may also be employed to represent many aspects of a computer system, for example, the Notebook metaphor (Fox and Gonzalez, 1989) is offered as an extension of the sorts of window manager systems discussed in more detail below. The metaphors of interest in this thesis are those that attempt to represent the facilities offered by a computer's operating system to support tasks such as file management. These interfaces are of interest as every user of the computer will employ them at some point, and they attempt to represent an artificial domain, that of the computer's storage facilities and operating system, with which most users will not be familiar beforehand.

2.1.1 WIMP Systems

Many of the systems discussed in this chapter, and considered in the rest of this thesis, are collectively termed WIMP systems. They are characterised by the use of Windows, Icons, Menus and a Pointing device (or Windows, Icons, Mice, and Pull-down menus, according to some interpretations of the acronym). An early discussion of the concept of windows may be found in Kay (1969), although many of the concepts and problems raised by windows date back through systems such as Sutherland's (1963) Sketchpad to early research in computer graphics. A window provides a view onto data or a data structure. Windows allow a portion of the data to be seen where the data is too large to be comfortably displayed on-screen in its entirety. Large graphics images and multi-page documents are examples of such data. Douglas Engelbart's rejection of the windows paradigm, as implemented in WIMP systems, and much of the discussion about post-WIMP interface design (for example, Van Dam, 1997) is due to the idea that "...WIMP interfaces are still 'marking interfaces' that in effect use 'digital ink' to make marks on digital 'paper' on a digital 'desktop'" (Bardini, 2000: 225). The complaint offered by critics is that windows are tied to these limiting metaphors. The power of, and the advance made

by, windows as proposed by Alan Kay was "...in part to eliminate the modality of applications. He also wanted to eliminate the distinction between operating system and applications but succeeded primarily in making the functioning of of the operating system visible in the form of the desktop." (Raskin, 2000: 141). The fundamental metaphors that critics argue underlie the window concept are not, however, faithfully implemented in real systems. Köhler (1987) observes that with many systems, notably text editors, the space in which data is displayed, which is viewed through the window, may itself depend on quite complex metaphors which must be recognised and interpreted by users. Only a portion of the entire display is visible within a window. In order to view the remaining portions of the large display scrollbars are often attached to windows. The scrollbar determines and represents the portion of the view currently visible in the window. The scrollbar can also give an idea of the size of the extract visible in relation to the size of the document as a whole, and the approximate position of the visible extract within the document. Smith (1987) discusses the notion of features which are literal to the metaphor employed by the system, or which are considered magical, in that they lie outside the metaphor yet increase the ease with which functions may be performed. Windows are powerful user interface features based on a metaphor, scrollbars have no analogue in the real world and are therefore magical, yet are demonstrably useful interaction objects. A more complex fundamental metaphor is discussed in detail in Section 4.4.

Smith (1977: 71) describes icons as "two-dimensional, visual, analogical, concrete descriptions of concepts." In the Xerox Star, icons simply denoted a closed window, whether this window provides a view onto the files in a directory in the file store, or it is a window employed by a currently active program. Icons are increasingly fundamental objects in user interfaces, they are usually visually *atomic* in that they have no internal structure, and hence may be employed as lexemes in human-computer dialogues. Unlike command-based user interfaces, the result of a command

may not merely be a description of the result of carrying out the command, but may provide objects which may be used *directly* in the user's subsequent task.

Menus are often implemented as a form of window, but with simplified mechanisms for scrolling through their contents. The menu, like its restaurant namesake, presents the user with a list of items from which a selection may be made. In user interfaces they are employed to present the user with a list of actions acceptable at the current point in the user's dialogue with the system. Users select and interact with the objects displayed on-screen using some form of pointing device. Light pens and datagloves are examples of such devices, the most commonly used are mice and trackballs, however, which convert the motion of the mouse, or the rotation of the suspended trackball, into changes in the position of an on-screen cursor. Input from buttons attached to the pointing device is used to select on-screen objects or cause operations on objects to be performed. A requirement of any pointing device is that a sense of *spatiomimesis* (Hutchins, Hollan, and Norman, 1986), immediate and appropriate feedback in the position of the on-screen pointer in response to movement of the pointing device, be perceived by the user. The importance of such system behaviour will be considered further below.

In the design of user interfaces which employ icons as a major component of the system, it has been noted (Gittins, 1986) that the model world of the system may be represented in terms of a useful metaphor by designing the icons according to a collective theme. A well-known metaphor employed in user interfaces, in which this is demonstrated, is the desktop metaphor. The desktop metaphor was devised for the user interfaces of the Xerox Alto and Star computer systems (Johnson et al., 1989) and was subsequently adopted by the Apple Lisa and Macintosh computers. Unlike some WIMP user interfaces, the desktop metaphor consistently employs icons designed according to the collective theme of an office environment.

2.2 The Desktop

The desktop builds on the ideas presented in the discussion of user interfaces which employ windows, icons, menus and some form of pointing device in employing a consistent metaphor in the design of the user interface. The path of development of the desktop metaphor has been a subject of much historical, and legal, argument. Johnson et al. (1989), Kay (1993), and Levy (1994) all provide details of the principal influences on the desktop's development. The desktop metaphor arose from work by the Xerox corporation into the design of systems to support the development of the electronic office. Smith et al. (1982a; 1982b) identified the option available to designers of employing metaphors in the design of user interfaces resulting in on-screen objects familiar to potential users from their everyday working environment. An example of an electronic desktop can be seen in Figure 2.1.

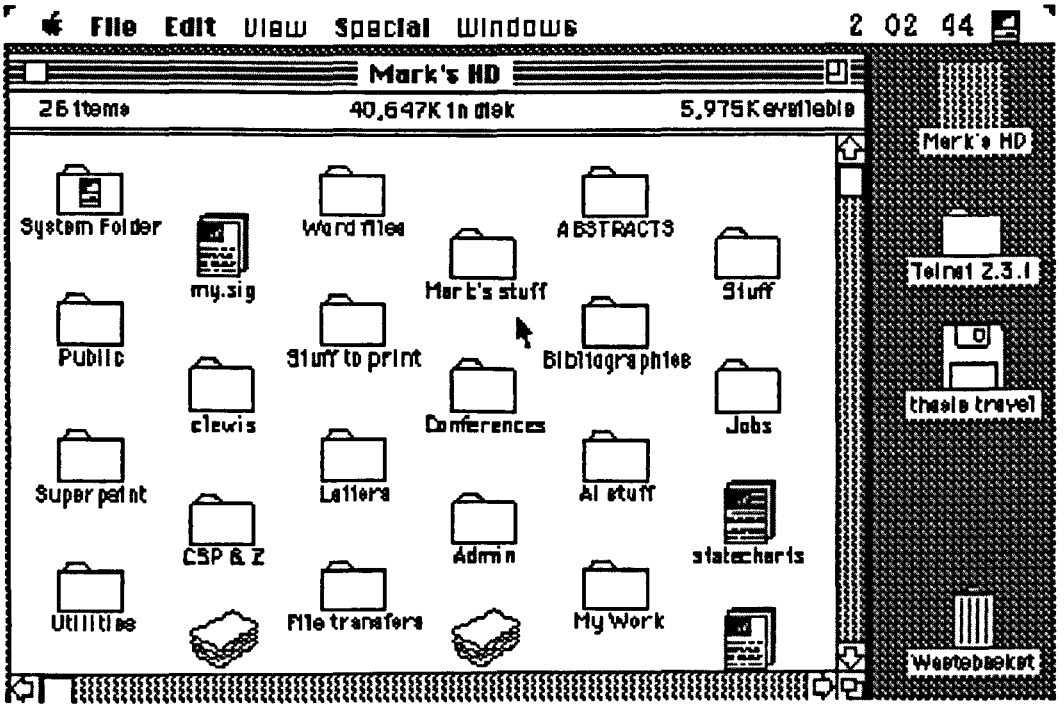


Figure 2.1 A Desktop

Smith and his colleagues, the designers of the Xerox 'Alto' and 8010 'Star' systems, the first commercially available systems to employ the desktop metaphor, recognised

that familiar analogies and metaphors may be used to introduce new concepts and functions to a potential user. The approach chosen by Smith and his colleagues was to create electronic counterparts of objects in the physical office with which the intended user population were familiar. In this way, the icons denoting text files are analogous to, and resemble, paper documents, directories on disks are analogous to folders, and electronic mail facilities are analogous to in and out trays. The design of the icons used in the Xerox 8010 Star's model world were the result of considerable design and testing effort (Bewley et al., 1983). Operations performed on objects are also analogous to operations that would be performed in the real world: filing a document requires moving it to the picture of a folder, whereas in the real world it would be carried to the physical folder itself.

Even in the design of the earliest desktop metaphor system, the Xerox Star, the system's designers appreciated the distinction between literal and magical features. The file storage system of the Star does not completely resemble real-world filing cabinets in that it adds a search mechanism which allows files or folders required by the user to be located without him or her having to browse the file structure tree. In the empirical study reported in Chapter 3 of first-time Macintosh users, such search facilities were used in preference to having to browse the file space. Search facilities are magical features, however, knowledge of a typical file organisation relies on the memory of a filing clerk, or on some some external catalogue. Certainly a request for the whereabouts for a file will tend not to produce the file ready for use, as is possible in direct manipulation user interfaces.

2.3 Rooms

The Rooms metaphor extends the notion of the WIMP user interface. It provides display structures that collect together related on-screen windows and addresses the particular issue of supporting task switching as a part of working practice and

computer use. Analysis of users' command histories (Bannon, Cypher, Greenspan, and Monty, 1983) has shown that users spend periods of time performing a certain task, but will interleave the commands employed in performing that task with the commands used to perform other tasks. Users this way spend periods of time on tasks punctuated by transitions and time spent performing other tasks.

In basic WIMP systems, the software tools employed in order to perform some task will each require one or more windows to be open on-screen. However, as users switch their attention from one sub-task to another, they are forced to switch their attention from one set of windows to another. The arrangement of windows on-screen has a great effect on the time it takes to switch attention from one window to another. The amount of *real estate* on-screen is often limited, and the number of windows visible at one time, or the size of the visible windows will be limited as a result (Billingsley, 1988). To overcome this space contention problem windows may be either tiled, or may overlap (Bly and Rosenberg, 1986). In a tiled window system, no window is obscured by any other window, however tiled windows may be very small. If a window is enlarged by the user in order to make its contents legible, then the other windows must be resized in order to remain visible on the screen. The recently developed *elastic windows* (Kandogan and Shneiderman, 1997) model is a space-filling tiled window, but one in which hierarchies of windows may be constructed to suit user roles and tasks, and in which operations, such as closing, may be performed on an entire hierarchy, not just a single window. Overlapping windows are more complex in that, in addition to having to be resized, windows not relevant to the current sub-task may have to be closed, or hidden behind windows that the user is interested in.

This problem of switching between windows has been likened by Card et al. (1985) to the use of virtual memory within a computer's operating system. Virtual memory allows a computer system to run programs which require larger amounts of physical

memory than the computer has available. This is achieved by storing the contents of memory locations that have not recently been accessed onto a secondary storage device and retrieving them into main memory when these data are required. Card and Henderson (1987a) employ terminology analogous to that of virtual memory in their discussion of the Rooms metaphor. They describe the requirement of having to ready an engaged tool, a software application used in performing some task, by manipulating its window, or by running the required application, as being initiated by a *tool fault.* If data located in secondary storage is needed in main memory, the terminology of virtual memory describes this as a *page fault.* Every time a user is forced to switch between windows on a computer screen, there is a delay as the user makes the required window visible. The Rooms metaphor attempts to minimise this overhead. The need to switch rapidly between tasks was noted by the Xerox Star's designers (Johnson et al., 1989) and led to tiled windows being employed in that system's user interface. As the number of tools required to perform the major task increases, the time spent switching between tools increases. In extreme cases, as with computer operating systems, the phenomenon of *thrashing* can occur; where users spend more time switching between tasks than they spend actually performing their tasks.

The design of the Rooms user interface is influenced by the observation that work conducted using a computing system is made up of phases of activity spent on particular sub-tasks using software tools punctuated by transitions to other sub-tasks performed using other software tools. The need to minimise transitions between sub-tasks and the software on which they are performed and hence reduce the time taken to complete the user's larger tasks is a particular issue addressed by Rooms. Central to the Rooms metaphor is the notion that all of the software tools engaged to accomplish a major task, such as reading electronic mail, are placed within one "room", or screen-sized work space. Tasks, however, may not be independent, an engaged tool may be used when performing two or more tasks. Also, it may be

desirable to have some tools, such as a clock, visible at all times in all rooms. An engaged tool may have a different role in one task to the role it has in another task, it should therefore be possible to adapt a tool to match the task. Figure 2.2 shows the relationships between tasks and engaged tools and Rooms and windows.

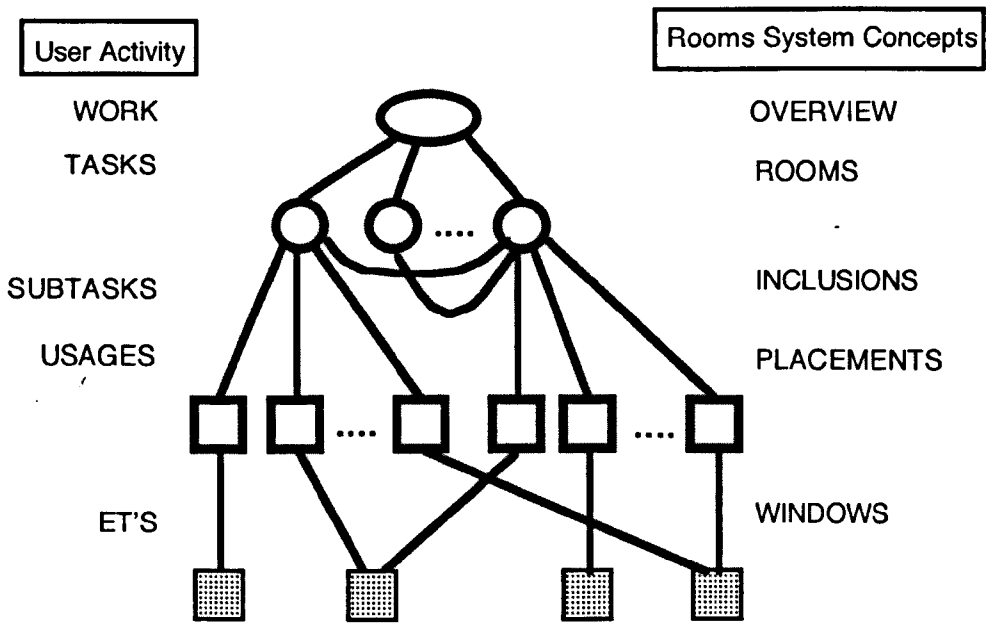


Figure 2.2 Relationships between tasks, engaged tools, Rooms and windows (Henderson and Card, 1986: 380).

A Room is a named screen-sized work space; in each room are the windows opened by the programs used to perform a major task. A Room containing a number of tools for reading and sending electronic mail, taken from Henderson and Card (1986: 224), can be seen in Figure 2.3. Each room also contains a number of icons resembling doors, these doors symbolise paths from one Room to another. To switch between tasks, the user clicks on the door to the Room that contains the engaged tools for the other major task.

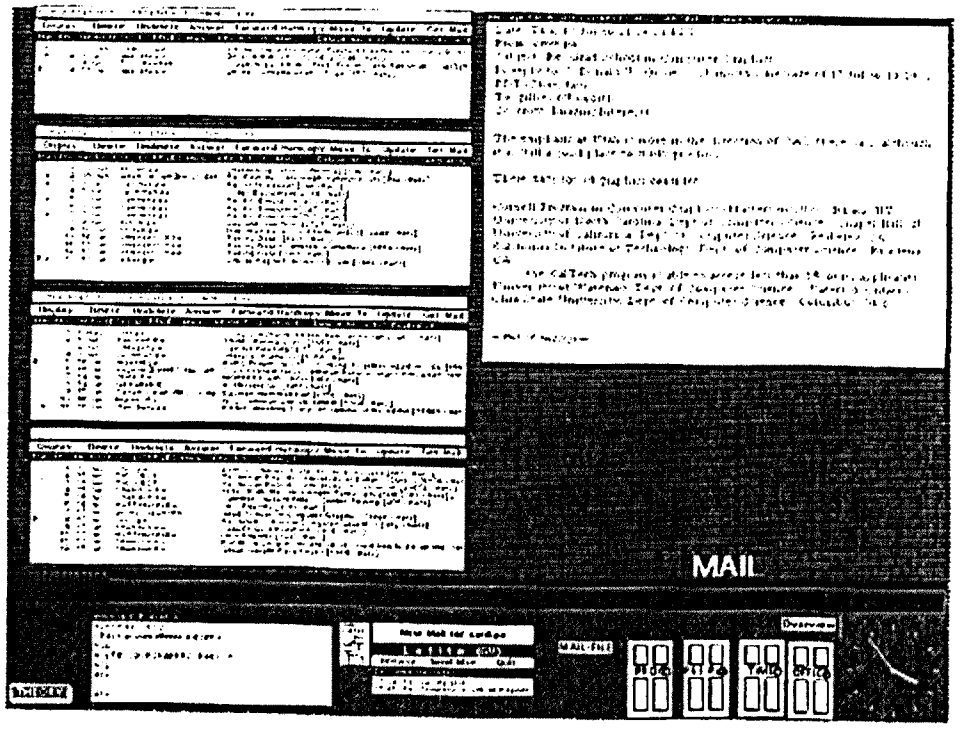


Figure 2.3 Mail, a Room for reading electronic mail

In case the user wishes to return to the Room that they entered the current Room from, Back Doors are provided. These special doors overcome the difficulty raised by most doors in the Rooms system that only permit one-way travel between Rooms, and help the user who may not remember the name of the Room they have just left. Card and Henderson (1987a) state that as the number of Rooms increases, the complexity of the interconnections between Rooms can create an electronic maze, for this reason two other mechanisms to aid the user navigate a network of Rooms are provided.

The first user navigation aid is a pop-up menu listing the names of all the Rooms in the network, from which the desired destination Room may be selected. The second solution is the Overview. The Overview displays a grid of pictograms of all of the Rooms currently in use arranged by the rooms' names in alphabetical order. To help the user find a particular window, window pictographs may be expanded to allow the user to browse through the windows in the entire set of Rooms.The paths between

doors and the Rooms they connect onto may, in addition, be superimposed on the Overview to show the web of interconnections between Rooms. These features, while improving the Rooms system, lie outside the basic Rooms metaphor, although the menu (as was mentioned in Section 2.1.1), and the use of a plan view of the network of Rooms in the Overview both rely on analogies and metaphors.

Other features that support users employ analogies that are closer to the central theme of moving between a number of inter-connected rooms. Users may wish to have an engaged tool, and the data associated with that tool, for example a text file and the editor used to prepare the file, accompany them as they move from one Room into another. The concept of *baggage* permits this. Baggage is simply the identification of tools that should travel with the user as they move into the next Room. If a number of tools are to travel with the user at all times, they are said to be placed in the user's *pocket* and appear in a Room within every Room the user visits. The notion of Room inclusion, having a Room contained within the current Room is the solution provided to the problem of defining the location and position attributes of tools that must remain constant across workspaces. If a change is made to any of the engaged tools in the collection, the change is propagated throughout the entire network of Rooms.

The Rooms metaphor provides a user interface which allows users to switch quickly between tasks without being delayed by the overhead of having to resize windows or to search for data files; the Rooms themselves, however, require a great deal of time to configure. In order to overcome this problem, Card and Henderson (1987b) devised the mail-order catalogue metaphor. The mail-order catalogue metaphor allows users to install and configure Rooms far more quickly than would otherwise be possible. Users may configure a network of Rooms by simply ordering pre-defined Rooms, Suites (small, pre-defined networks of Rooms) and engaged tools from the catalogue. By employing the catalogue, the user can define a network of

Rooms which will be available instantly when the user starts the machine, users can also use a previously unused software application with far less difficulty than if they were using the basic Rooms system.

2.4 The Alternate Reality Kit

The Alternate Reality Kit (henceforth ARK) was developed by Randall Smith (Smith, 1986). ARK shares many of the features of the systems, and is influenced by, the same systems that influenced the development of the systems mentioned discussed above. Its name, for example, follows from David Canfield Smith's (Smith, 1977) Pygmalion system's provision of an *alternate reality* for supporting creative thinking in its users. ARK's principal influence is the Smalltalk programming language and the principal aim of the Smalltalk environment to be a system for developing microworlds, interactive simulated environments.Motivation for ARK followed from Smith's observation that students of physics demonstrate difficulties in understanding the abstractions encountered in physics. Studies conducted using ARK, which shall not be discussed further, have shown that ARK is helpful in overcoming students' difficulties in understanding Newtonian and relativistic physics. An example of an ARK simulation, taken from (Smith, 1987: 62) can be seen in Figure 2.4.

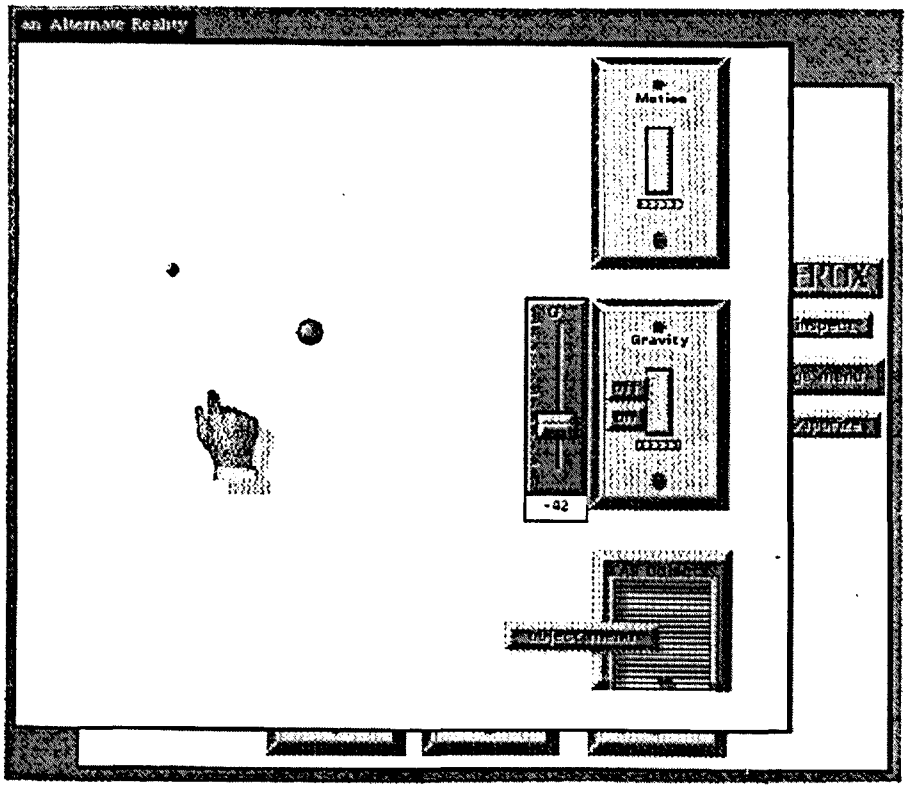


Figure 2.4 An ARK simulation of bodies moving under mutual gravitational attraction

ARK simulations are constructed by providing access to object-oriented programming in Smalltalk-80 for non-expert programmers. ARK provides a number of pre-defined objects from which simulations may be constructed, prototypes of these objects are all held in the warehouse (shown in Figure 2.5) from where the instances of objects required for a particular simulation may be retrieved. ARK provides other on-screen objects that are used to alter variables encapsulated within a simulation object. Slider switches are used to specify numbers, they allow values of properties associated with an object to be easily altered. Buttons (shown in Figure 2.6) are the means by which users communicate directly with objects. Buttons contain a simple command to be applied to an object and are invoked by being picked up using the hand pointer and dropped onto the object. If the user wishes to remove an object from a simulation, the user drops the "vaporize" message onto that object, the object will then disappear. Message passing is the mechanism by which

objects communicate with each other, message passing can also be thought of as the means by which users interact with on-screen objects as suggested by Card, Moran, and Newell (1983). This concept will become important when human-computer dialogues in a new user interface design are considered in Chapter 6.

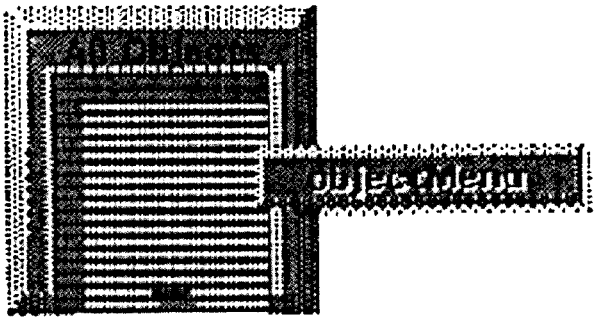


Figure 2.5 The ARK warehouse (Smith, 1987: 65).

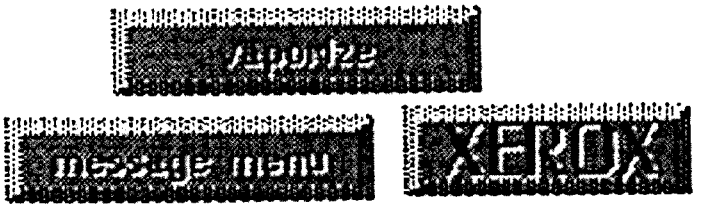


Figure 2.6 ARK buttons (Smith, 1987: 65).

Message boxes provide a general message passing facility, a message box consists of the name of the message it sends and a plug that connects to the object which is to receive the message. If the object is to return a value as the result of being sent a message, the message box will contain a region in which the result is displayed, for example, an object might be "asked" for its mass and this value would be displayed within the message box. Representatives (shown in Figure 2.7) often appear as an object that contains text describing the object being represented. Representatives allow instances of any Smalltalk-80 class to be represented and used within an ARK simulation. Interactors (shown in Figure 2.8) allow users to manipulate physical laws within a simulation. Interactors define an object's behaviour, or define constraints that apply between a number of objects, for example Newton's inverse square law of

gravitational attraction, and they also maintain a list of the objects within a simulation subject to that constraint. The power of an interactor lies in users being able to adjust its attributes in the same way that they can adjust the attributes of other objects. The user could, for example, reduce the gravitational constant, or switch it off entirely.

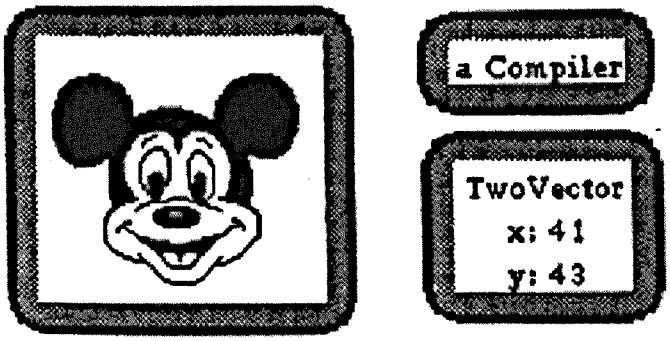


Figure 2.7 ARK representatives (Smith, 1987: 65).

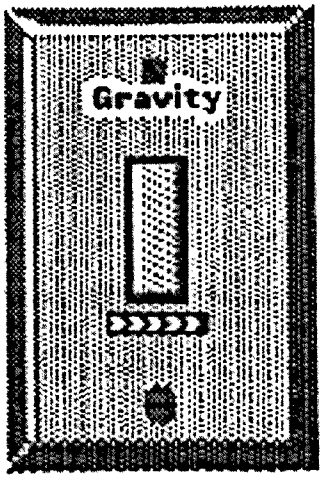


Figure 2.8 An ARK interactor (Smith, 1987: 65).

Users interact with all ARK simulations using the *hand*. The hand, like other on-screen pointers, is used to select and manipulate the on-screen objects. ARK permits several alternate reality simulations to run at the same time, each within its own window. A number of simulations could, for example, show the same set of objects interacting with different sets of physical constants as a way of comparing how

changing the value of one constant affects the simulation. The hand (shown in Figure 2.9) is not a part of any of these realities, it exists in a meta-reality. ARK complicates the simple physical world metaphor with the concept of a reality structure. The alternate reality simulations are self-contained, but all lie on one plane of reality. The hand exists in a meta-reality where it is free to move without being subject to any influences from the alternate realities, and from where it casts a shadow in the reality below. Any object picked up by the hand is taken into the meta-reality and the objects left behind behave as if the object were no longer there. Buttons attached to objects also cast a slight shadow signifying that they intrude into the meta-reality. An object's position in the reality structure is meant to aid novice programmers by eliminating the confusion between editing and execution, the object's appearance denoting its current role.

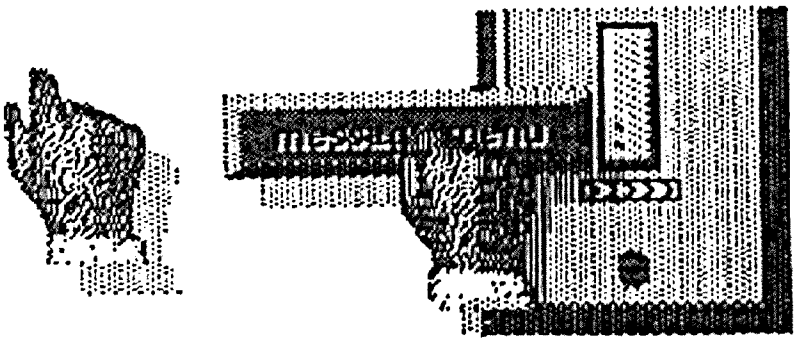


Figure 2.9 The ARK hand (Smith, 1987: 65).

Although the ARK is based on a physical-world metaphor, some features of the ARK's interface, such as attaching buttons to objects, would be very difficult to achieve if they were activities literal to the metaphor. Actions such as attaching buttons to an object by simply dropping the button onto the object lie outside the physical-world metaphor and are considered magical in Smith's (1987) distinction. The use of magical features that lie outside the metaphor has implications when users are learning to use the ARK. Smith (1987: 62-63) says:

> "... one of the lessons of ARK is that the literal aspects of the interface are often obvious while magical capabilities are harder to learn. In ARK, the time to explain the basics is actually measured in seconds. Every piece of added magic is relatively 'expensive' because it requires its own explanation: it does not 'come for free' as it does when the user realizes there is a physical metaphor."

2.5 Metaphor and Non-visual Representations

In the sections above the use of metaphor in visual forms of representing software objects in a small number of computing systems was considered. These systems all rely on the visual modality to communicate the system state and in the depiction of the metaphor employed. In all of the systems discussed above, sound is either absent or limited to a few simple indications that an event of some sort has occurred. And while the mouse, or some equivalent device, is used to point to and select objects and operations on objects, these systems cannot be said to employ the haptic channel to communicate system feedback, or to communicate the user's intentions to any great extent. In Section 2.6, we consider the role metaphor plays in systems that employ the haptic channel to a greater extent than in the systems described above. In this section, we consider the role of metaphor in systems that employ other modalities to a larger extent than in what are typically deemed metaphor-based systems.

We are unaware of the olfactory channel, the user's sense of smell, being currently employed to communicate information about the state of an interactive system (except in the case of some severe hardware failures). While Morton Heilig's Sensorama arcade rides, which are cited as early immersive reality systems (Rheingold, 1991), would, in one ride, blow the smells of combustion fumes at the rider of a virtual motorcycle, the rides themselves were not interactive. The user was

simply a passenger on a ride filmed earlier and projected onto eyepieces giving a 3D display. In Smith's (1996: 231) terms, however, as the user can "see through" the projection to the *actual* events and objects it presents, the display does not stand in metaphorical relation to other events and objects and so this system does not require further consideration in this thesis. Sound, however, as a means of communicating information about the current state of a computing system, is worth some discussion.

2.5.1 Auditory Icons

It is claimed that other than its use in computer games, sound still tends to be neglected as a means of conveying information in computer systems. Where sound is used, if at all, in most systems, it is restricted to "beeps" and other simple warning sounds. Gaver (1986) noticed this neglected modality and outlined an approach that uses sound to convey a great deal of information about a computer system to the user. This approach, termed *auditory icons*, uses caricatures of naturally occurring sounds to represent both conceptual objects and dimensional data within the computer system to the user. Auditory icons are mentioned as they can evoke metaphors in the way that they communicate information. The auditory icon approach is not concerned with the proximal stimulus, meaning the dimensions of sound such as pitch, loudness and duration that describe the variations of air pressure near the ear, but rather is concerned with the distal stimulus, with the physics of the source of the sound.

Gaver (1986: 168) described the information that might be conveyed by an auditory icon saying:

> "One can imagine how a single sound could be used to give information about a file arriving in a message system. The file hits the mailbox, causing it to emit a characteristic sound. Because it is a large

message, it makes a rather weighty sound. The crackle of paper indicates a text file - if it had been compiled program, it would have clanged like metal. The sound comes from the left and is muffled: The mailbox must be in the window behind the one that is currently on the left side of the screen. And the echoes sound like a large empty room, so the load on the system must be fairly low."

If a sound is to be used to represent a source of information, the mapping between the information and the representation, the relationship between the source and the sound, must be considered. Gaver (1986) identifies three mappings between source and sound; nomic, termed iconic in (Gaver, 1989); metaphorical; and symbolic. Symbolic mappings have an arbitrary mapping between the information and its representation, they rely on social convention for meaning, examples of symbolic mappings include sirens and telephone bells. Earcons, "which are short, rhythmic sequences of pitches with variable intensity, timbre and register" (Brewster, Wright, and Edwards, 1993: 222), another form of auditory feedback that have received some attention, have only a symbolic mapping to the object, location, operation, or interaction that they denote. Nomic, or iconic, mappings in auditory icons depend on the physics of the source of a sound to convey meaning, an example is the auditory icon described above representing a file being placed in a mailbox.

Metaphorical mappings rely on similarities between the represented and the representing systems to convey meaning. A metaphorical mapping may either be a structural mapping where similarities between the structure of two symbols or objects are exploited, or it may be a metonymic mapping, where a feature of the object is used to represent the whole object. Gaver (1986) gives the example of a hiss being used to represent a snake as an example of a metonymic mapping. Other metaphorical mappings rely on the notion of temporal progression of sound and the events the sounds stand for, or on the notion of a dimensional metaphor, "... in which

one ordered dimension is used to represent another" (Gaver, 1986: 171). An example Gaver (1986) gives of the use of a dimensional metaphor is of the change in pitch of an object at different heights.

The mappings between the represented object and the representation are not distinct descriptions, it is possible for an auditory icon to employ a mapping that lies between two of the classes of mappings described. If a metaphor is weak, or is poorly understood, then the mapping becomes increasingly symbolic. Also, nomic mappings depend on models of the source events for understanding, as models become more approximate, the result becomes more like a metaphor. Nomically mapped auditory icons also depend in some sense on metaphors, the icon's mapping will, Gaver (1986: 172) claims, "... be nomic to some event in the model world presented to the user, not to underlying events in the computer itself." Auditory icons have, to date, been implemented within two important systems, the SonicFinder (Gaver, 1989) and SharedARK (Gaver, Smith, and O'Shea, 1991), these are briefly discussed below.

2.5.2 SonicFinder

The SonicFinder (Gaver, 1989) augmented the desktop metaphor of the Apple Macintosh Finder user interface with auditory icons. The auditory icons were added to the Finder system to provide auditory information whenever the user interacted with an object on the model desktop. For example, if the user clicked on the visual icon representing a file and dragged the icon across the screen, the user heard the sound of the object being hit (clicked on) and a scraping sound as the object was dragged. Further sounds were added to typical actions that can be performed within Finder; the actions of opening windows and scrolling the contents of a window, for example, had auditory icons associated with them.

Gaver (1989) realised that this use of auditory information is redundant, the Apple Macintosh had been used successfully for some time without sound being employed to the extent it was in the SonicFinder. His claim, however, was that learning and remembering the system is aided by this redundant information, by these *confirmatory sounds* (Gaver and Smith, 1990), and that users' perceived senses of direct engagement with on-screen objects should be enhanced — although no studies exist to substantiate these claims.

2.5.3 SharedARK

SharedARK is a multiuser version of the Alternate Reality Kit (described in Section 2.4). In ARKola, a simulation implemented in SharedARK, auditory icons were used to convey information about hidden processes in a soft-drinks bottling plant (Gaver, Smith, and O'Shea, 1991). The ARKola bottling plant is made up of a number of interconnected component machines, but only a few machines can be seen on a user's workstation at any one time so much of the operation of the plant will be invisible. Auditory icons were used to convey information about these invisible machines.

The use of auditory icons in such an application is likened to the way in which some people, especially trained and experienced mechanics, can determine the status of a machine with which they are familiar depending on the noise that the machine is making. If there is a fault within a machine, it is assumed to cause a characteristic noise which can aid diagnosis of the fault. Within the ARKola factory, users are able to tell if the factory as a whole is running well from the noises made by the separate component machines. If there is a fault in the running of the factory, users are able to tell which machine to examine from the characteristic noise made by the faulty machine, for example if the bottle storage area is being overfilled, the sound of

breaking bottles can be heard. Studies of collaborating users undertaken using ARKola (Gaver, 1991; Gaver and Smith, 1990; Gaver, Smith and O'Shea, 1991) demonstrated that for some states of the system the use of auditory icons assisted users in determining which tasks they should attend to next. Some sounds were less effective than others, the *absence* of critical sounds was in particular not regarded with the urgency it should have been. The results, however, lead Gaver and his colleagues to claim that auditory icons are useful for communicating semantic information, rather than just for event notification or communicating simple status or mode information, which are the typical uses of sound in interactive systems.

2.6 The "Reality" Metaphor and New Interaction Styles

The systems described above are landmarks in metaphor-based user interfaces. All of these systems are confined to the (physical) desktop and to running on a conventional workstation (we shall ignore personal digital assistants for now). The *"reality" metaphor* is a term coined by the researchers working on the Wearable Computers project at the MIT Media Laboratory. The "reality" metaphor describes the presence of *both* real physical objects and computer-generated artefacts in the user's visual field. A growing movement in computer science is the design of systems that are mobile, ubiquitous, or a natural part of the environment. We are required to consider such systems, not just because of their growing importance, but also because of the role metaphor plays in the design and understanding of them. We shall not review all the systems that can be termed as applying the "reality" metaphor, but shall briefly discuss illustrative examples of the different interaction styles that fall under this heading. We discuss in greater detail the fundamental metaphors claimed to be the foundation for many systems designed according to the "reality" metaphor.

Ubiquitous computing, as the term suggests, is concerned with making computing systems a part of the everyday environment. This can be achieved by introducing computing machinery into artefacts that have previously not contained computing systems, such as LEGO[1] bricks (Resnick et al., 1996) or office whiteboards (Stafford-Fraser and Robinson, 1996), or by making computing systems smaller and mobile, as in the form of PDA's (personal digital assistants). Whereas in ubiquitous computing systems, the computing system and the physical artefact occupy the same object, be it a LEGO brick, doorknob, running shoe, and so on, in augmented reality systems the external world provides implicit input into a computing system. In augmented reality the user interacts with a real world augmented by computer-generated information. Examples of such systems include repair assistants (Bass et al., 1997) where instructions can be displayed *over* the image of the *actual* object being repaired. A CCTV camera mounted in the computing device, or head-up display provides an input source to the device and possibly relays the image of what it is looked at to the user. Devices with such an arrangement of camera and display are described as employing a *magnifying glass* metaphor (Rekimoto and Nagao, 1995). This term is a true metaphor (the "information is detail" metaphor) in that it describes a system, but not one where the image of the world seen by the camera is magnified in the display and more detail can be seen. Instead,, the image is magnified in terms of the *information available,* additional information being supplied by the computing device, not the world itself.

Some systems employ the reality metaphor to support the task domains supported by the workstation-bound user interface metaphors discussed above. The desktop metaphor is typically used to support tasks that are performed in an office setting, the desktop metaphor, however, mirrors the environment into which it is introduced, it is not fully a part of it. Documents must be printed if they are to be stored in

---

[1] LEGO is a trademark of LEGO Systems, Inc.

physical filing cabinets, but a printed document is unavailable for manipulation in the electronic domain. The DigitalDesk (Wellner, 1991), for example, overcomes these problems, by use of image projectors, cameras linked to image processing software, and a touch- and gesture-sensitive physical desk surface. In the DigitalDesk, electronic and physical documents have equal status within the system, images of electronic documents may be projected onto the desk's surface and real documents may be scanned and an electronic version of them created. Systems developed as part of the Tangible Media project at the MIT Media Laboratory (Ishii and Ullmer, 1997; Ullmer and Ishii, 1997) demonstrate a similar equality of physical and electronic objects within the representation and embodiment of the task domain.

2.6.1 Optical Metaphors

The tangible user interfaces developed by Ishii and his colleagues are based on metaphors of light, shadow, and optics, which are claimed to be "particularly compelling for interfaces spanning virtual and physical space." (Ishii and Ullmer, 1997: 240). The activeLENS system, an arm-mounted flat-panel display is described as being modelled in both its form and function as a jeweller's magnifying lens, the same notion drove the design of the passiveLENS, a simpler transparent glass surface onto which the metaDESK display projects information. The metaDESK greatly extends the use of optical metaphors.

The metaDESK concept, depicted in Figure 2.10 (MetaDESK itself is shown in Figure 2.11), is an effort to integrate both computer and physical worlds. Via the desktop metaphor, aspects of the physical world are emulated in the 2D model world implemented by a PC. The metaDESK concept simultaneously attempts to physically instantiate windows, icons, menus, handles, and control metaphors back into the real world (denoted A in Figure 2.10), as well as expoiting affordances of

real world instruments and artifacts made obsolete in the development of the personal computer (denoted B in Figure 2.10).

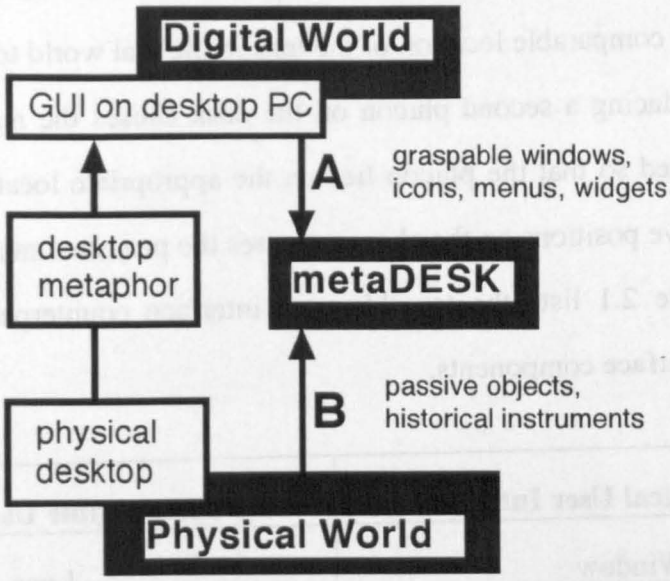


Figure 2.10 The metaDESK concept (Ullmer and Ishii, 1997: 224).

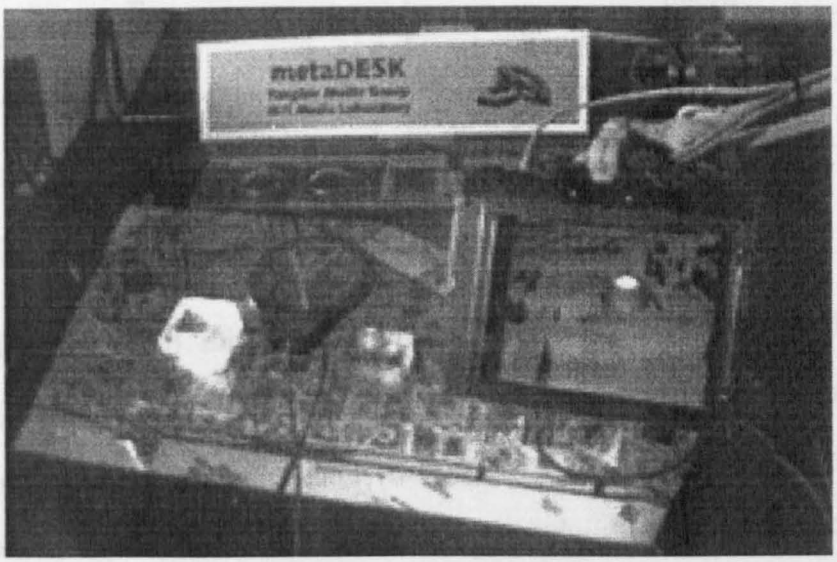


Figure 2.11 MetaDESK (taken from Dourish (2001: 45)

The metaDESK concept is illustrated by the prototype application *Tangible Geospace*. In this system, small physical replicas (collectively termed *phicons*) of landmarks found on the MIT campus can be placed on the surface of a desk onto

which a map of the MIT campus rotated and translated appropriately to match the orientation and location of the phicons is back-projected. Viewing the desk through the activeLENS will cause an appropriate 2D projection of the 3D scene that could be 'seen' from the comparable location of the lens in the real world to be displayed in the lens' panel. Placing a second phicon on the desk causes the map to be scaled, warped, and rotated so that the phicon lies on the appropriate location in the map. Moving the relative positions on the phicons causes the projected map to be adjusted accordingly. Table 2.1 lists the tangible user interface counterparts of common graphical user interface components.

GUI: Graphical User Interface	TUI: Tangible User Interface
Window	Lens
Icon	Phicon
Menu	Tray
Handle	Phandle
Widget	Instrument

Table 2.1 Physical Instantiation of GUI Elements in a TUI

In terms of optical metaphors, phicons are linked with the notion of "digital shadows". As illuminated objects cast shadows, so phicons cast digital shadows that project information as to their virtual contents. Thus Ishii and Ullmer suggest that a suitably modified torch (flashlight) can be used to project different wavelengths of virtual, or semantic, light onto the desk. One form of light might render physically constrained shadows of the physical building, while another might cause funding for the faculty to be rendered.

The ambientROOM system provides information not only to 'foreground' perception, as with the metaDESK, but also to peripheral perception through ambient media,

light, shadow, sound, air and water flow. Where ambient information needs to brought into the foreground for closer attention, the ambientROOM provides phicons that act as sources of the ambient information, which may be moved into the proximity of an information sink, such as a loudspeaker where the information can be suitably rendered. The use of optical metaphors in all of these systems is justified by Ishii and Ullmer's (1997: 240) claim that:

> "Perhaps the most compelling aspect of the optical metaphor is its seamless consistency with the physics of real space. By not only invoking but also obeying the optical constraints metaphorically imposed on our physical interface prototypes, we are able to maximize the legibility of interface in our creations. People know what to expect of a flashlight, know what to expect of lenses. By satisfying these expectations, we can truly realize truly seamless 'invisible' integration of our technologies with the physical environment."

We shall consider these claims further in Chapters 6 and 9.

2.7 Conclusions

This chapter served to review a number of existing user interface designs which employ metaphors in order to attempt to represent a large part of the underlying computer system. While the systems described above revolutionised, and continue to revolutionise, the usability of computing systems and make them accessible to a far larger number of users, the use of metaphors in the designs of their model worlds is not a perfect solution to the problem of improving system usability. In the following chapter we examine user interface metaphor in general, and the problem of

understanding and interacting with computing systems in terms of metaphors and analogies. In particular, we survey the difficulties that user interface metaphors pose for the user.

Chapter 3

An Empirical Study of First-time Macintosh Users

"I feel I am beyond metaphorical assistance."

— 'Cher' in "Clueless", the television series.

In the previous chapter a number of important and influential computing systems and user interface designs that implement user interface metaphors or which rely on metaphor comprehension in order to be understood and used successfully were reviewed. These systems helped to make the personal computer the pervasive technology that it is in many parts of the world. They also helped make the computer accessible to a wider range of users than other visions of personal computing might have seen come about. The design principles of these systems can be contrasted, for example, with Douglas Engelbart's "bootstrapping" concept and NLS technology (Bardini, 2000), which was intended for use by knowledge workers who were expected to invest tens of hours in the initial training period. Many of the systems described in the previous chapter have not been subjected to usability testing that would make the case for their usability and usefulness compelling. Alternatively, usability testing might have been undertaken, but the results of such testing might have been withheld for commercial reasons. The study reported below, for example, seeks to copy one whose results were withheld for some time at the behest of the

company that sponsored the original work (John M. Carroll, personal communication).

The systems described in the previous chapter all use or rely on metaphor for their understanding and use. A survey of the HCI literature, however, finds that metaphor is also a source of users' difficulties. In this chapter we report on a small study of first-time users of the Apple Macintosh to test the notion that metaphor is always advantageous in interface designs. This study also seeks to examine the findings of Carroll and Mazur (1986) who undertook an empirical study of the Apple Lisa (the forerunner of the Macintosh) and found a number of usability faults that were not resolved by the use of the desk top metaphor. Indeed they found (their study is described in more detail in the following chapter) that the use of metaphor can be a source of users' difficulties.

In Carroll and Mazur's (1986) study, a small number of subjects from the staff of IBM were recruited who used an Apple Lisa for weekly sessions lasting between two and three hours to undertake a medium-scale project and report. Due to constraints imposed on the study reported below, our study examines a much shorter period of initial use of the DESKTOP metaphor. Like Carroll and Mazur, though, we also found that the DESKTOP metaphor used in the user interface, while it is an improvement over command-based interfaces, was a source of users' difficulties. These difficulties, among others, that are due to the use of metaphor are explored further in the next chapter.

3.1 Overview of the study

3.1.1 The Subjects

Seven subjects, recruited from the staff of the Open University, participated in the study. The subjects were selected on the basis of availability, and were all paid a nominal fee. All of the subjects were familiar with IBM PC compatible computers, and, over the group, were familiar with spreadsheets, database software, terminal emulators, C language compilers and word processing packages. All of the subjects were users of the Microsoft Windows graphical user interface to MS-DOS. As subjects were only available during their lunch break, sessions were planned so significant progress could be make in no longer than an hour.

3.1.2 Methodology

Subjects were supplied with a copy of the Apple Macintosh manual (Apple, 1990) and a blank, pre-formatted floppy disk. Subjects were also given a list of short exercises to perform (shown below). These exercises were designed to be similar to the sorts of tasks that users of the Apple Macintosh would perform every day, and were designed to force those subjects who attempted them into using particular parts of the system. The subjects were not obliged to perform the exercises, although they all attempted them. The subjects were then free to begin to investigate the Macintosh system. Subjects were asked to "think aloud" as they worked, otherwise they were free to work as they wished. No advice was offered as the subjects worked. When they could see no solution to their problems, the session was considered to be at an end and advice was given. Notes were taken and the subjects' screen activity was videotaped and their speech recorded. The time subjects spent working ranged between forty minutes and an hour and ten minutes.

3.1.3 Tasks Performed by Subjects

Participants were provided with the list of tasks below, as mentioned they were under no obligation to attempt them, although all participants did, with mixed success. Participants were told before attempting the first exercise:

> Try to do as many of the exercises as you can. Don't worry if you cannot complete an exercise or if you haven't completed an exercise before time runs out.

The instructions for the tasks were as follows:

> **Exercise 1**
>
> Write down the names of the files in the folder **Experiment 1**
>
> **Exercise 2**
>
> Delete all of the picture files from the folder **Experiment 1**
>
> **Exercise 3**
>
> Copy the application program in the folder **Experiment 1** to the floppy disk provided.
>
> **Exercise 4**
>
> Find the file **Mary's lamb**, open it, and add the line **the lamb was sure to go** to it. Save the file and make a duplicate copy of it. Place the duplicate copy of the file on the floppy disk.

Exercise 5

Eject the floppy disk from the computer.

3.1.4 Caveats

Although Carroll and Mazur's (1986) study was taken as the starting point for this study, this study differs in a number of respects. Firstly the machine used was not an Apple Lisa, instead an Apple Macintosh SE/30 running the System 7 revision of the operating system was used. This machine was chosen as one of the particular aims of the study was to investigate users' understanding of the mechanism which allows the computer to have a number of programs active at one time and which allows the user to switch between these programs.

The internal floppy disk drive of the machine used was faulty, and an external floppy disk drive had to be used instead. The Apple Macintosh supports three methods of ejecting floppy disks from machine. The "proper" way is to drag the icon of the disk to be ejected to the trashcan, the automatic mechanism within the disk drive will then eject the disk. The second method, which is commonly used when copying files from one floppy disk to another, is to select the **Eject Disk** option from the **Special** menu or to use the ⌘**E** shortcut. Again the automatic mechanism will eject the disk, but a *greyed out* image of the disk's icon will remain visible on the desktop. At some point in the future the system will request that the disk be replaced so that any outstanding or final operations on it may be performed. The third method, which is only recommended should the machine crash and there is no other way of retrieving the disk, is to push a rod (such as a straightened paper clip) into the hole to the right of the disk drive, this action will manually eject the disk. The external drive, unlike internal Macintosh disk drives, had an eject button, which acted in the same way as the ⌘**E** key combination. Whilst this study used an atypical hardware configuration,

many flaws in the subjects' understanding of the computer system were brought to light as a result.

The principle differences between this study and Carroll and Mazur's are the time students used the system for, and the tasks they were asked to perform during that time. Subjects could only give up an hour of their time, so many aspects of the Macintosh user interface could not be encountered by them. The exercises that the subjects were set (see above) were considered reasonable for the length of the session, although those subjects that completed the exercises did so in less than the time available. The subjects in Carroll and Mazur's study were available for two three-hour sessions, hence were able to perform a much more complex task using the Lisa, and many more aspects of the Lisa's interface were encountered by their subjects.

3.2 Observations

3.2.1 Using the Manual

The subjects were all provided with a copy of the user manual supplied with the Macintosh computer (Apple, 1990), none of the subjects, however, found it to be of much use. Subjects 2 and 7 were the only participants to attempt to make much use of the manual, the other subjects who used the manual did so only as a last resort (three of the subjects did not refer to the manual at all).

One problem that arose was the layout of the manual; subjects would find the page number of a topic they were interested in, but would find that the contents of that page addressed a different topic. Because of the method of instruction used by the manual, subjects had difficulty searching from the page containing information that they did not want to the information that they did require. Subjects would be

presented with the sequence of steps to be followed to perform some specific task, but they were unable to apply this action sequence in the particular context of the task they wanted to perform.

Using the manual to search for information on a specific topic was complicated by some subjects re-phrasing the terminology of the desktop metaphor into terminology with which they were familiar, this re-phrasing was also evident when users attempted to perform certain other tasks. This meant that subjects would re-phrase a problem in the list of exercises into terminology they knew, and then search the index for the familiar term rather than the correct term. The subjects would then be unable to obtain the correct information, if they were able to obtain any useful information at all.

3.2.2 Using the On-line Help Facility

The computer system used for this study was equipped with an on-line help facility called Balloons. When the user runs this program, a small speech balloon appears next to an object that the user points to using the mouse. The balloon contains a small piece of text that describes the object and its possible uses. A typical speech balloon can be seen in Figure 3.1.

Subject 2 made particular use of these balloons, using them initially to gain information about all of the on-screen objects, then, as the session progressed, she used them to gain information about single objects that she had not encountered before. On-line help such as this proved initially very helpful. As the session progressed, however, some shortcomings became apparent. Firstly, speech balloons had only been defined for a limited number of objects, and had not been defined for objects used by a number of application programs. Thus there was no information about objects that were new to the subject. Secondly, Subject 2 in particular became

frustrated by the text that appears in the speech balloon being a constant piece of "canned text", the information does not change according to the particular state of the machine or in relation to the task the user is trying to perform.

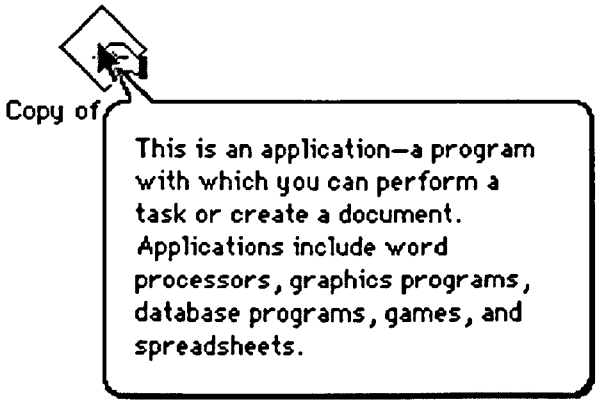


Figure 3.1 An on-line help speech balloon

3.2.3 Interpreting the Desktop Metaphor

The subjects were all familiar with the Microsoft Windows user interface, so they already possessed many of the basic skills needed to use the Macintosh. Some of the subjects' existing skills, however, were particular to the Windows environment and interfered with their attempts to learn the Macintosh. The aspect of the Macintosh that caused most problems when subjects attempted to apply their existing knowledge is not one which is covered by the desktop metaphor, hence discussion of this will be delayed until Section 3.2.4. Some problems with the desktop metaphor did, however arise.

Most of the problems observed by Carroll and Mazur were caused by the terminology used to describe the system, this also caused problems in this study. The notion of an application file caused problems for some subjects, they simply were not sure what was meant by this term. Even when they had double-clicked on an

application's icon and could see the program running, some subjects (especially Subject 6) could not associate an application with a runnable program. The concept of a folder caused some confusion, most of the subjects at some point remarked that they assumed by the new term "folder" that the more familiar term "directory" was meant. Folders caused a problem for Subject 7. He did not relate the icons that appeared in the window that opened when he double-clicked on a folder's icon to the files contained within that folder, he assumed them to be more folders, even though none of the files had a folder-shaped icon attached to them.

The terminology used to describe the Macintosh and its user interface created more serious problems when users tried to perform certain tasks. The exercises listed in Section 3.1.3 were all phrased in the terminology used in the system documentation, but as users attempted to perform these exercises, especially in the case of Subject 3, the terms used were translated into more familiar terms . However, in reformulating the description of the task it was sometimes then impossible to perform the task. When, for example, Subject 3 came to delete a file on the hard disk, she saw the task as one of "erasing" a file. She was then forced to use a number of elaborate strategies in order to perform the act of "erasing" when she could find no information on "erasing" files in the manual. For example, she carried out a lengthy search of the options listed on the pull-down menus and even opened the file in the hope that she would find an option within the application that would be capable of erasing the file.

The trashcan was also found to create problems for the subjects. Users seemed not to notice that a desktop metaphor was being used as far as the trashcan was concerned. Only Subject 1 knew that files could be deleted by dragging them to the trashcan, and she admitted that she had been told about this before the session. The other subjects tried to apply their knowledge of Microsoft Windows and went on, often lengthy, searches for a delete option on a menu. Most subjects resorted to the manual

after this searching proved fruitless. It occurred to Subject 4 that dragging files into the trashcan might delete them, which he then tried. This provoked him to remark:

> "It's that easy is it? ... I suppose if I'd taken the time to read the manual I'd have found that out."

Once subjects had learned to delete objects, some still did not infer some other tasks that could be performed using the trashcan, Subject 2, for example, asked:

> "How do I retrieve things from the wastebasket?"

Only one subject discovered that disks can be ejected from the disk drive by dragging their icon to the trashcan, but she discovered this information by stumbling across it in the manual. All of the subjects initially used the eject button on the drive. This, as was mentioned in Section 3.1.4, causes the user to be frequently prompted by the system to re-insert the disk. These frequent requests prompted only one user to ask if there was another way to eject disks, the others were seemingly content, and did not notice that the system had not completed any outstanding operations on the disk before ejecting it.

The design and use of icons on the Macintosh were not as successful as might have been assumed. For example, only one subject was able, from looking at the icon alone, to deduce that the icon shown in Figure 3.2 represented a picture created using a graphics software package. The remaining subjects used combinations of two strategies to determine the contents of the file. One method used was to simply double click on the icon of every file installed on the system's hard disk that they were interested in discovering the contents of. This had the effect of running the application used to create the file, or ran the application itself, and the subjects would then decide the nature of the file from what appeared on the screen. The other

method was to use the **Get Info** option on the Finder's **File** menu. This has the effect of displaying information such as the size, creation date, and the nature of a file, whether it is a document or application and so on.

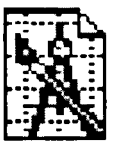


Figure 3.2 Icon denoting a file produced by SuperPaint[1].

Again, similar strategies were adopted to determine which was the application file to which the exercises referred, (the shape that all application icons should have, shown in Figure 3.3, is described on the very first page of the Macintosh manual, but went undiscovered). This is surprising as all of the subjects had experience of a graphical user interface, it suggests that an association is learned between the icon shape and the file and the software package used to create that file. Subjects were unable to decide which icon denoted a picture because they had not learned the association, but it had been assumed before the study began that subjects would be able to infer the nature of the file from the design of the icon.

Figure 3.3 An application program

[1] SuperPaint is a trademark of Silicon Beach Software, Inc.

3.2.4 Basic User Interaction

All of the subjects were familiar with the Microsoft Windows user interface, hence they already possessed much of the knowledge required to use the Apple Macintosh system. This knowledge, however, also interfered with their attempts to learn the new system. An example of this was the subjects' use of the mouse button. To pull down a menu within the Microsoft Windows system, for example, the user has only to press and release a mouse button. On the Macintosh, however, the menu remains visible only while the mouse button is pressed, if the user releases the button while an option on the menu is highlighted, then that option is selected. The subjects all required several attempts at pulling down a menu before they learned that they needed to keep the mouse button pressed, once they had learned this, most of them had no further problem using menus.

Problems that seemed to bother all of the subjects, but which posed particular problems for Subject 5, were the inconsistent results of selecting options on menus and the results of pressing buttons on windows. Mostly, operators are *overloaded*, that is, the same operator is able to perform the same function on a number of different objects. These are the *generic operations*, such as *open*, *close*, and *print* that are discussed by Rosenberg and Moran (1985) and which were invoked by keys on the Xerox Star keyboard labelled with the operator's names. Some operators are, by contrast *polymorphic*, operators with the same name have different semantics depending on which object they are applied to. On the Macintosh system used in this study, the same operator name, or button on a window, provided a number of different operations but no information was provided by the display to tell users which operation would occur. The close window button (shown in Figure 3.4) is an example of this, sometimes clicking on this button causes a window to close, but the application continues to run, sometimes the application is closed down. Without

checking the list of active programs on the **Finder** menu, it is often difficult to tell which operation has occurred.

Figure 3.4 Close window button

The **Open...** operator on the **File** menu caused particular problems for Subject 5, again due to inconsistent assignment of operators to names and buttons. The version of Microsoft Word installed on the machine used in the study had the behaviour that if the user pulled down the **File** menu and dragged the pointer to the **Open...** option — but not release the mouse button — a sub-menu listing readable files in the current directory would appear. Subject 5 tried to apply this knowledge at the desktop level of the Macintosh's interface. This caused her to be unable to perform a seemingly simple task. Exercise 4 asks the subject to find a file that contains a few lines of text and to add a further line of text. Most subjects, possibly because of the way in which they use Microsoft Windows, did not browse the hierarchy of files on the hard disk, rather they used the **Find** option on the **File** menu. Subject 5, again rather than search for the file, remained at the root folder of the file space tree and attempted to discover the contents of each of the sub-folders. She did so by selecting the folder of interest, highlighting it, and choosing the **Open** option from the **File** menu. Rather than release the mouse button she waited for the sub-menu to appear, when it did not, she falsely concluded that the highlighted folder was empty. A possible explanation for this is the subject perceiving the keyboard shortcut **⌘O** on the menu as **#0**, meaning that the number of items in the folder is zero, that the folder is empty.

Making the underlying state of a computer system visible, or easy to infer, has been stressed as important when considering what makes a system usable and when considering how successful mental models are formed. A large number of problems that arose were due to information about the system's state not being made as explicit as it should have been. The System 7 revision of the Finder user interface allows a number of programs to be resident in memory at one time, if there is sufficient main memory, and the user can then switch between these programs. If the user clicks on the **Finder** menu icon (see Figure 3.5) a menu listing the programs currently resident in memory will appear, the user may then make one of these programs active by selecting it from the menu.

Figure 3.5 Finder menu icon

We described above how the **close window** button was thought by Subject 5 in particular (although this problem was encountered by all subjects to some extent) to close down an application rather than simply close the window. This effect was compounded by the occasional use of the close window button actually shutting down the application. This result is interesting as the system is still said to be *predictable* (Dix, 1991). After the user closes a window, the menu bar at the top of the screen states that the system will behave as if it is still running Microsoft Word, say, because the system *is* still running Microsoft Word. This on-screen information was ignored, however, by users who believed instead that they had achieved their goal of closing down the application program. Roast and Harrison (1994) discuss *templates*, areas of the display that contain information about the state of the system relevant to the user's goals when performing tasks. In this situation it seems that the desktop appearing from beneath the closed window confirms the user's hypothesis

that the application has been closed down, and that the information contained in the menu bar (a far smaller section of the screen) is ignored. A further source of information that would inform users that the application was still active, the list of resident applications, is hidden in the **Finder** menu and by the time subjects discovered this menu, they were unable to relate the list of programs to their command history. They did not know that the list of applications presented by the **Finder** menu was a list of applications that they had previously run and which had not been closed down. Subjects seemed to persist in the belief that they had closed down applications even when problems arose when they attempted to run other additional applications. One subject was informed by the system that there was insufficient RAM to run the application that she wished to, but it did not occur to her that other programs were idle and taking up space in main memory that could be otherwise used by shutting down some unused programs.

The problems that arise due to subjects not being aware of programs still being resident in the computer's memory are compounded by the hiding of information about the state of the underlying machine, but one can understand why. For users to successfully use this version of the Finder user interface, called MultiFinder, they need to be aware of the program switching mechanism and need to be aware of some mechanism which can focus its attention on a single program and run it. Users need to be aware of a (however vague) notion of a hidden processor, but this lies outside the scope of the DESKTOP metaphor. In providing this mechanism for allowing uses to switch between a number of programs, the advantage to the user of employing a metaphor, making the underlying computer system invisible, has been lost. Also, in trying to reconcile the program switching mechanism and the DESKTOP metaphor, information in the display needed to make the system usable has been hidden.

3.3 Conclusions

The results of this study concur with those of Carroll and Mazur's study (which is described in more detail in the next chapter). Subjects were found to have problems with basic user interaction with the system as well as in comprehending the desktop metaphor. Many of the problems with the basic aspects of the system appeared to be a result of subjects' prior knowledge of a different graphical user interface being applied to the new system. A number of these problems, however, were compounded by the design of the Macintosh user interface itself, and by information that would have helped subjects learn a correct model of the system being hidden.

The documentation supplied with the Apple Macintosh appears to rely on rote learning of the skills needed to use the system successfully. This contrasts with the active learning approach adopted by the subjects, who would try any seemingly useful approach to achieving some goal before "resorting" to the manual. Indeed, three subjects announced that they were totally confused and left the session early after trying to perform comparatively simple exercises, when the information they required was easily obtainable from the manual, and should have been deducible from the desktop metaphor.

Subject 3 gave some hint as to how learning this sort of interactive system could be made more successful. She remarked that the method adopted by the study where subjects worked alone without human advice was not her preferred way of learning, she preferred to have an adviser on hand should she need someone to answer her questions. Certainly a facility, human (such as a work colleague or helpdesk advisor) or otherwise (such as an intelligent help system or agent), able to offer some form of context sensitive advice would have been useful to the subjects and help, if offered at the right time would probably have prevented the subjects who left early from doing so. Context sensitivity was something lacking from the balloon on-line help, the

subject who made great use of the balloons bemoaned the use of canned text in them and would have preferred more specific help.

The introduction to this thesis hinted that while metaphor is a widespread and useful technique in user interface design, it can also be a source of users' difficulties and usability problems. In the next chapter drawbacks and usability problems arising from the use of metaphor-based user interfaces will be described in more depth. A number of these drawbacks were observed during the small empirical study that was reported in this chapter. In order to go on to present new user interface designs based upon a fuller account of metaphor as it applies in HCI, we must examine the difficulties that metaphors can give rise to. This is the task of the next chapter.

Chapter 4

Drawbacks to Employing Metaphors and Analogies in Interactive User Interfaces

"So the question is, will they see the metaphor?"

— Arthur Miller.

4.1 Introduction

In Chapter 2 a number of computer systems which exploit metaphors in order to support file management, support object-oriented programming, provide a user interface for application programs, and provide mechanisms for switching between application programs were discussed. Although metaphors and analogies are suggested as a partial solution to the problem of designing usable software, metaphors often cannot account for aspects of a software system, and sometimes the use of a metaphor can create new usability problems while solving others. In this chapter some of the drawbacks of employing metaphors in user interface design are presented.

Carroll, Mack, and Kellogg (1988) identify three strands in research in the use of metaphor in human-computer interaction. They make a distinction between *operational* and *structural* approaches to metaphor, and the *pragmatics* of metaphors in use. This distinction is adopted in the consideration below of some of the drawbacks in adopting metaphors in user interface design. In the following

discussion, the seeming convention of considering analogy and metaphor to be synonymous in user interface design is adopted, but this will be challenged later.

4.2 Operational Metaphors

Operational metaphors are applied in an educational context in order to make the process of teaching some concept simpler. Metaphors are provided by the teacher or instructional material and their value is judged by the learning gain that results over circumstances where no explicit metaphor is employed. Operational approaches to metaphor, according to Carroll et al. (1988), therefore attempt to provide examples of "good" and "bad" metaphors for certain concepts.

The work of Richard Mayer is often cited as an example of employing operational metaphors to teach and explain computing systems. Mayer demonstrated the value of teaching programming in the BASIC programming language with relation to a concrete analogical model of the underlying system. The model taught to some of Mayer's subjects is described in Mayer (1976). Input to the system is said to pass through a physical window in the form of cards with some data written on them. Output from the system resulting from the execution of WRITE commands, is written on the topmost available line on a pad of paper. The flow of execution through a program is monitored by the commands making up the program being listed on a card, and the current command being pointed to by an arrow. The current values of the program variables is written into boxes on a chalkboard, each box is labelled with the name of the corresponding program variable. As the variable's value is altered, the learner erases the current value from the relevant box on the chalkboard and writes in the new value. This model of program variables has been shown by Burstein (1986) to be insufficient to prevent some learner errors arising, however.

Mayer (1981), as well as considering the teaching of programming languages, also provides a concrete model for a computer's file storage system and file management command language. This again is presented in the form of concrete analogies, a writing pad is used for output from the system, and a chalkboard is used for representing the list of variables used by the program. Program instructions are listed on a pad and a pointer is used to indicate the current instruction, as in the case of the set of analogies used to teach BASIC programming. The file management system differs from BASIC alone in having a filing cabinet used as an analogy for the storage of a set of files. Each file exists in a separate drawer in the cabinet, and is said to be made up of a number of records on cards. Files are read by removing the cards from the filing cabinet drawer and placing them in an IN tray on a desk. As the file is processed, some record cards might be discarded, these cards are placed in a DISCARD tray. Cards that are altered and are to be saved are placed in a SAVE tray, from where they are returned to the appropriate drawer in the filing cabinet.

Mayer's studies of employing such analogies showed demonstrable positive effects on learning if learners were given such models of the system before reading conventional user manuals. Mayer suggested that the analogies provide a framework into which the new information contained in the manuals may be assimilated. As will be discussed further in Section 4.5, properties associated with the metaphorical, or analogical, explanation may not match properties associated with the system to be explained. In the case of the explanation provided for the file management system, Mayer is forced to tell learners that only one drawer of the filing cabinet may be open at a time. The reason for this is that only one file in the computer's storage system may be accessed and altered at a time for reasons that are well known in the design of database and operating systems. Learners may be aware that more than one file of a typical real-world filing cabinet may be opened, hence the filing cabinet analogy does not provide a perfect match for the storage of computer files. The learner may demand some reason for the mismatch , which will have to be given in

terms of the actual properties of the file management system. Real filing cabinets may indeed only permit one drawer to be open at a time, but it is difficult to explain the prevention of file corruption through mutual exclusion of file updates in terms of the weight of open filing cabinet drawers causing the cabinet to topple over. While Mayer reports positive outcomes from providing students with a concrete analogy for the BASIC language, the evidence for the usefulness of such *advance organisers* is mixed. A similar study conducted by Foss et al. (1982), in which learners of a new system were given a model very similar to that given by Mayer in order to explain the file save facility of a text editor demonstrated a far less clear advantage. Experiments conducted by Payne (1988), by contrast, show advantages in learning device semantics and command abbreviations when metaphorical instruction is provided.

Rumelhart and Norman (1981) present a model of learning in which new knowledge structures in the form of schemata (Bobrow and Norman, 1975), are developed initially by applying existing schemata which may be employed analogously to the problem at hand. The example they give is of drawing a pentagon in the Turtle graphics system of the LOGO programming language, which is described as an analogous procedure to drawing a square. In this example, the structure of the schemata which is employed in the operation of constructing the command to draw a square stays the same, but the loop parameter used to specify the number of sides is altered, and the internal angle between sides of the intended polygon is adjusted. Rumelhart and Norman go on to examine their model of analogical use of schemata in the context of the result of a study of users learning to use the UNIX text editor Ed. They suggest instances of the system's commands which can be employed using schemata analogous to schemata representing understood commands, and they suggest that evidence from protocols taken during the study support the view that learners do employ such mechanisms in learning. Problems arose, however, when learners reasoned analogously from the known results of some known commands to

the expected effect of other unknown commands. The result of issuing a command to print a line, for example, is for the contents of that line of the document to appear on the display. Learners reasoned that the result of deleting a line would cause that line to disappear from the display, which it did not. Rumelhart and Norman's suggestion is that the mental models that learners bring to the learning of the system play roles in the analogies they apply when using the system. The role of mental models in system learning will be discussed in Chapter 5. In order to overcome the problems raised by the system not behaving as the learners' analogical reasoning predicted, Rumelhart and Norman were forced to give the learners further information about Ed more appropriate to its use than the analogical predictions made. Rather than give information about the system inappropriate to the learners who had little knowledge of computers, a solution similar to Mayer's operational accounts of BASIC was adopted, and the system was described in terms of a 'secretary' model, a 'tape recorder' model, and a 'card file' model. The secretary model is used to account for the mixing of commands and text supplied to Ed by the learner. It is often found, however, that when a system displays some intelligent behaviour, users often bestow more intelligence upon the system than it actually possesses. Describing Ed in terms of an intelligent system, a secretary, led to users behaving as if the system should be able to recognise typed input that should be interpreted as commands when in Ed was in the append mode (where typed input is merely appended to the text file). The tape recorder model overcomes this problem, termed the *append-mode trap*, by providing the model of a system which records everything faithfully until explicitly ordered to stop recording. This model, however, cannot account for delete functions that Rumelhart and Norman described using the card file model. Each line of text is thought of as being typed onto a record card. Deletion commands remove relevant cards from the stack that makes up the entire document. Such difficulties are not the only ones encountered when using text editors, as will be discussed in the next section. The operational metaphors given by Rumelhart and Norman in their study, it may be noted, impose on learners the need to recognise which metaphor is to be

employed to help describe the system and their current task where metaphors overlap to describe the same aspect of the system.

4.3 Structural Approaches to Metaphor

Hall (1989: 43), from a survey of existing work, identifies the following components of a process model of analogical reasoning:

1. *Recognition* of an analogical source.
2. *Elaboration* of an analogical mapping between source and target.
3. *Evaluation* of the elaborated analogy.
4. *Consolidation* of information generated while using an analogy.

The operational metaphors discussed in the previous section address the *recognition* component of Hall's analysis where, given a target domain and a set of source domains, the problem is to find a promising set of candidate sources. This set may be then employed in a tutorial context or further refined to produce the most suitable candidate with which to solve problems (Carbonell, 1983). The *elaboration* and the *evaluation* components address the problems of finding a mapping, the analogical inferences, and mapping preferences between a source and target domain, and evaluation of a mapping given a source and target domain and analogical inferences, respectively. These components are the consideration of structural approaches to metaphor and analogy.

Hall (1989) describes the elaboration component of analogical reasoning as the problem of finding a mapping and a set of analogical inferences given the target and source domains and mapping preferences. Existing accounts of finding mappings between the metaphorical base, or *source*, domain and the previously unfamiliar target domain mostly rely on knowing the structure of both domains.

A number of approaches to the representation of analogical source domains and a destination target domain, the method of determining a mapping between source and target, and the evaluation of the mapping have been proposed. Rumelhart and Abrahamson (1973) model similarity between domains as a distance metric between points (which denote concepts) in a multi-dimensional space. Such models cannot, however, account for asymmetric similarities considered central to understanding metaphors. An example of asymmetry cited by Tversky (1977: 328) is that people say that "an ellipse is like a circle" not that "a circle is like an ellipse". In Tversky's model similarity matching is made according to the function:

$$s(A,B) = F(A \cap B, A - B, B - A)$$

meaning that the similarity of A to B is expressed as a function, F, of three arguments; $A \cap B$, the features that are common to both A and B; A – B, the features that belong to A but not to B; and B – A, the features that belong to B but not to A. Tversky's theory is extended to address non-similar domains, or metaphors, by Ortony (1979), but Ortony's model is unable to make judgements as to the quality of a metaphorical mapping.

Mac Cormac (1985) proposes a model of metaphor in which concepts are members of fuzzy sets. This model is employed in the study of linguistics in an attempt to understand metaphors in natural language sentences. Sentences are thus regarded as metaphorical; non-metaphorical (literally truthful); or epiphors, which "... involve outreach and extension of meaning through comparison"[1]; and diaphors, where "...

---

[1] P. E. Wheelwright (1962) *Metaphor and Reality*, Indiana University Press, Bloomington, Indiana: 72. Quote reproduced from Indurkhya (1992: 77).

similarity need not be obvious and comparison explicit"[2]. Mac Cormac's model allows a once metaphorical utterance to become a 'dead' metaphor, and a part of ordinary language over time by its membership of non-metaphorical fuzzy sets increasing and its membership of the fuzzy sets which denote forms of metaphor decreasing. Mac Cormac also provides a useful means by which analogy and metaphor may be discriminated between and also reconciled with his model. Metaphors, being statements which are not literally true, but which are stated as true, create an *emotional tension* in the reader which forces him or her to search for attributes of the metaphorical source which may be applied to the target domain.

Much of the study of analogy and metaphor has been concentrated in linguistics and natural language communication. The study of analogy in reasoning, problem solving and planning is a growing field and has given rise to a number of models and representations of source and target domains, a number of these being developed from work in artificial intelligence. The most often employed model of analogy in human-computer interaction and the learning of computer-based domains is Gentner's structure-mapping model (Gentner, 1983). In the structure-mapping model, the source and target domains are both represented as a number of objects, every object has a number of attributes associated with it, each denoted as a single argument predicate taking the object's name as the argument. Relations are also said to apply between objects, these are represented as predicates taking more than one object as arguments. Second- and higher-order relations may also be defined which take first- and higher-order relations as predicates. Domains described in terms of objects, attributes and relations may also be represented graphically in a graph structure. Figures 4.1 and 4.2 show the domain structures which the analogy "the atom is like the solar system" can be made between.

---

[2] P. E. Wheelwright (1962) *Metaphor and Reality*, Indiana University Press, Bloomington, Indiana: 74. Quote reproduced from Indurkhya (1992: 77).

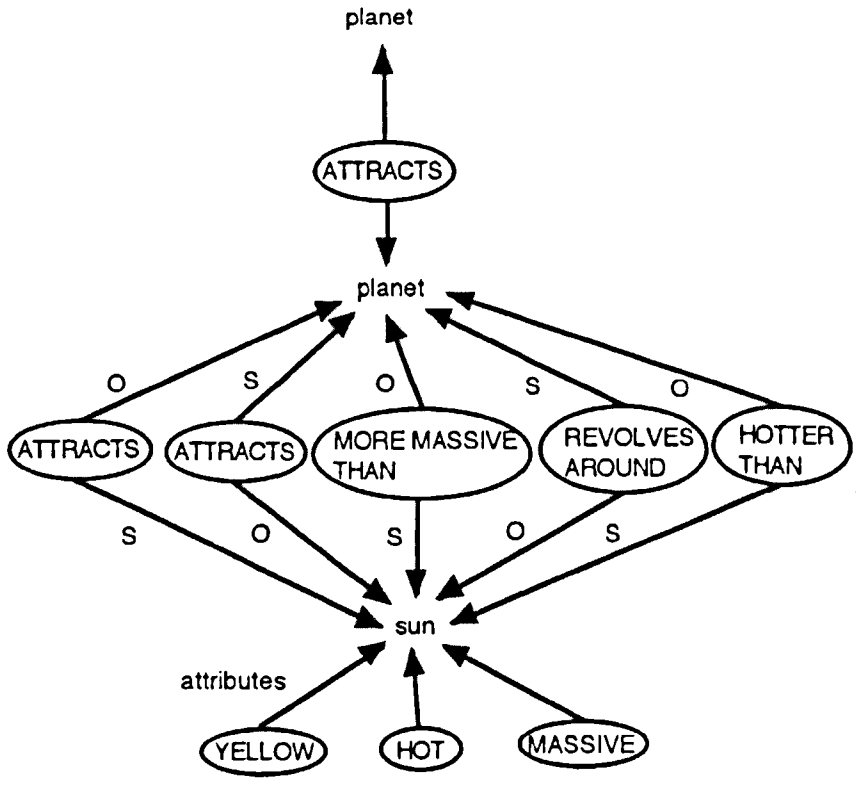


Figure 4.1 Domain model of the solar system (Gentner, 1983: 160).

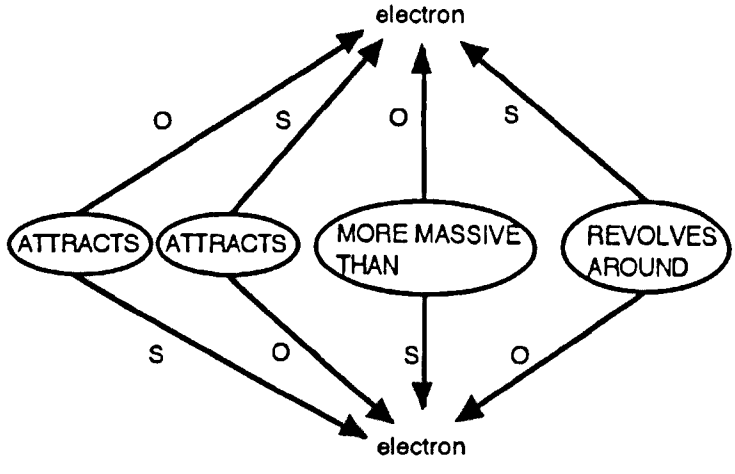


Figure 4.2 Domain model of the structure of the atom (Gentner, 1983: 160).

Provided with suitably represented source and target domains, the structure-mapping model provides a method for mapping from the source to the target and evaluating the mapping. The mapping is achieved by first discarding the attributes of objects in the base and target domains, and by attempting to preserve and match the relations between objects in both domains. Deciding which relations are preserved is achieved

by the *systematicity principle*, according to which a predicate that is part of a "... mappable system of mutually interconnecting relationships is more likely to be imported into the target than is an isolated predicate" (Gentner, 1983: 163). Interconnected predicates may be identified from higher-order relations in the domain description. The types and numbers of predicates mapped from the base (source) to the target give an indication of the success of the metaphor being employed, and place the relationship between the base and target domains on a continuum from literal similarities to analogies. Within the structure-mapping model metaphors are treated in a similar way to the approach described by Mac Cormac (1985). Suitable attributes, objects (some of which may not be the initial domain representation of the problem or utterance), and relations between objects, need to be identified and considered in the mapping process. This process may be seen in Winston (1980) where facts about a domain may be increased or generalised to aid the analogical mapping process.

Where the structure-mapping model performs the mapping from source domain to target domain based on syntactic structures, and exact similarity of higher-order relations in the representation, in the ACME model (Holyoak and Thagard, 1989) *semantic* components in the two domains are matched. In addition, mapping between the two domains in ACME can be less precise than in the structure-mapping model. A judgement as to the best mapping between domains is made by comparing the level of an excitation function produced by each of a set of computational elements which each evaluate a potential mapping. Several plausible mappings may be generated, the best mapping is chosen according to the element that achieves the highest excitation level. A similar approach is realised in the Copycat system (Mitchell and Hofstadter, 1990). Keane's IAM model (Keane, Ledgeway, and Duff, 1994) also generates and evaluates potential mappings in parallel, but imposes realistic constraints on time and memory limits so as to better emulate actual human performance. IAM, like ACME or its predecessor ARCS (Thagard et al., 1990), also

relies heavily on background knowledge and semantics in making mappings between domains. The MAC/FAC model (Forbus, Gentner, and Law, 1994), while retrieving possible analogues in parallel, still employs the Structure Mapping Engine to perform mappings.

4.4 Structural Approaches to Metaphor and Learning of Computer-Based Systems

Douglas and Moran (1983) studied a number of computer-naive people learning the text editor EMACS. Learners were provided with the operational metaphor of a typewriter by the teacher, and in some cases were seen to employ this analogy without prompting. Special attention was paid to the structure and semantics of operations from the typewriter source domain, and the semantics of operations in the EMACS target domain. . Rather than construct a mapping between the typewriter domain and EMACS domain representations, Douglas and Moran instead constructed a problem space (Card, Moran and Newell, 1983). This described both the EMACS system and the effect of EMACS commands (operators) on the system and the current document when attempting to perform tasks and achieve goals. Operators applicable in the typewriter domain were then mapped into the EMACS domain. This interpretation of the analogical reasoning process allowed Douglas and Moran to build a taxonomy of errors which occur when the *operators' semantics* are wrongly applied in the EMACS system. Douglas and Moran suggest that 62 out of 105 errors (59%) observed in protocols obtained from novice EMACS users are explicable in terms of wrongly applied operators from knowledge of typewriting.

The cursor keys give rise to particular errors, for example, the visible effect of the <Cursor Right> key was mistaken for the visible effect of the space-bar, although the result on the document was different. The destructive effect of the <Backspace> key also caused learners problems. The insertion of an invisible character in the text at

the point where the <Return> key is pressed requires users to understand a more complex model of the space into which text is typed than the simple sheet of paper that would be used with a typewriter (Köhler, 1987). The model of the space which EMACS and other text editors employ is the *saw-tooth sheet.* Figure 4.3 shows part of a text file that has been highlighted (or selected) by lassooing a number of lines of text by pressing the mouse button down and dragging the pointer over the text. Unlike a sheet of paper where we might expect the selected region of space to extend to the right hand edge of the page, each line ends with the usually hidden end-of-line or line break character (depicted as the ¶ symbol in Microsoft Word) giving the saw-tooth shape to the selection.Rather than advance to the edge of a sheet of paper, the effect of pressing the <Cursor right> key when the cursor is at the position of the invisible *Carriage Return* character is to cause the cursor to advance to the first character of the next line. Mistaking the effect of the space bar and the <Cursor right> command, which is *passive* and has no effect on the actual text, at this point would give rise to very different effects when navigating or altering a document.

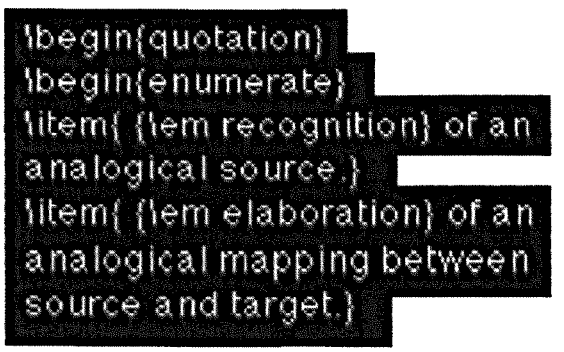


Figure 4.3 Highlighted text placed on a saw-tooth sheet

Allwood and Eliasson (1987) report that in a similar study to Douglas and Moran's in which a database system was studied, only 6% of the learners' errors could be accounted for in terms of misapplied analogical mappings. , Although they suggest this figure may depend on the type of system being considered. Allwood and Eliasson also re-consider Douglas and Moran's results in terms of a greater number

of categories in which learners' errors could be placed. Rather than place all errors that could be accounted for in a category of errors caused by the use of a typewriter analogy, other types of analogical error were accounted for in additional categories. These additional categories included *priming analogy* errors where a command was initially used correctly, but misused later, and *anticipation analogy* errors, where a command used matched a command that would be correctly used in a sub-task, but not in the context of the current task. With these additional categories, only 14% of learner's errors were said to be caused by use of a typewriter analogy, but errors caused by all types of analogies accounted for 60% of errors noted. Allwood and Eliasson also suggest that 68% of all errors were due to analogies if inefficient uses of commands caused by analogical reasoning in system use were classified as errors.

Examining the structure of the target domain of the desktop metaphor, Benyon et al. (1990: 30) notice that:

> "... it is common practice to include an icon of a dustbin on the 'desk'. Not only does this contravene our expectations as to where to find dustbins (on the floor), but also the interface dustbin has other functions apart from its conventional use as a container for discarded objects. For instance, the dustbin is often the place where disk icons are put in order to eject the disk from the disk drive. This implies that one has to 'throw away' a disk in order to retrieve it! Such an apparent contradiction can cause conceptual problems to first-time users since it is easy to think that the contents of the disk will be discarded when the disk is placed in the dustbin."

In addition to the inconsistent way in which objects such as the wastebasket behave, Carroll and Mazur (1986) report that rather than being able to employ the DESKTOP metaphor to understand the computer system beneath, users often find it

difficult to understand the desktop itself. This is discussed further in Section 4.5 where the pragmatics of metaphors and system learning are considered.

Structural approaches to the analysis of metaphors used to teach computing concepts, or in user interface design, are applied so that the worth of a particular metaphor may be determined from the quality of the mapping between the source and target domains (Carroll and Thomas, 1982). Carroll and Mack (1985: 39) suggest that exploring the use of metaphor according to operational and structural approaches ignores the "goal-directed learner-initiated learning *process* though which metaphors become relevant and effective in learning." Carroll and Mack propose an *active learning* process by which people learn unfamiliar systems in an open-ended way using the metaphorical features of a system to generate initial hypotheses and operations that are refined with increased exposure to the system, and greater experience using it. Evidence is mixed as to whether structural evaluation of analogies may be used to decide on the better analogy for use in a particular domain, or whether an active learning approach should be assumed, and that the choice of analogy for a domain matters little in eventually understanding the domain.

Chee (1993) employed Genter's structure-mapping model of analogy to produce instruction materials to teach BASIC programming with a good analogy, a weak analogy, and in a control case, no analogy. According to the structure-mapping model, best results should be obtained with a good analogy, less good results obtained with no analogy, and worst results with the weak analogy. Chee found that the most successful learning was achieved with the best analogy, as determined by the evaluation mechanism in structure mapping. Contrary to expectations, the results for the weak analogy and no analogy cases were not substantiated, although the results were not significant, they were suggestive of the expected result. Chee suggests that the weak analogy might not have been as weak as it could have been

made, the criteria by which an analogy is evaluated within structure mapping are linked and altering one criterion may affect another, which may improve the overall usefulness of the analogy. Galloway (1993), however, supports the active learning approach. Galloway attempted to teach a number of computing concepts to groups who were taught using either weak or strong analogies (as determined by a structure-mapping evaluation). Whether weak or strong analogies were employed had no effect on the eventual learning outcomes demonstrated by subjects in the two groups. Galloway therefore suggests that both weak and strong analogies facilitate learning the previously unfamiliar domains.

4.5 The Pragmatics of Metaphor

The final current approach to the consideration of metaphors in user interface design is to consider the pragmatics of metaphors in use in real systems. Carroll, Mack, and Kellogg (1988) observe that using metaphors inevitably involves dealing with incompleteness, mismatches, and composite comparisons, yet they suggest that metaphor mismatches can prove useful. The Alternate Reality Kit (discussed in Section 2.4) is a useful tool, for example, because it allows the student to gain greater understanding from confronting their naive physics with accurate physical models encoded as ARK simulations. ARK is also of benefit by providing a safe environment in which to experiment with the objects under study and their attributes. In this section, some of the issues surrounding the pragmatics of learning and using user interface designs are discussed. Pragmatic approaches to metaphor examine the use of metaphor-based systems in plausible real-world situations in which the system might be used. In this way, more information may be gathered about the success of the metaphor than may be obtained from a structural analysis alone.

4.5.1 WIMP Systems

The Apple Lisa, a forerunner to the Macintosh, was one of the first commercially available systems to employ the DESKTOP metaphor in its user interface. Carroll and Mazur (1986) conducted a study of learners using the Lisa for the first time This section briefly discusses the problems users encountered when attempting to learn this system, and the problems caused by the adoption of the desktop metaphor. Studies such as Carroll and Mazur's and the study described in Chapter 3 demonstrate that these systems are often more difficult to use and learn than proponents of metaphor-based systems suggest.

4.5.2 Instruction

The Apple Lisa was supplied with an on-line tutorial entitled LisaGuide. LisaGuide will not be discussed in detail as it is specific only to the Lisa computer, but the methods it employs to provide instruction to novice users are worth describing. The LisaGuide tutorial is made up of a number of on-line lessons, each consisting of a number of exercises designed to make the user familiar with some aspect of the system. These exercises are to be performed one after another, and are to be performed by rote. Users are unable to structure the sequence of exercises, even though, as one of Carroll and Mazur's subjects found, the exercises seem pointless and simple to master in their given order. The LisaGuide teaches simple skills at first, such as use of the mouse and mouse buttons, and goes on to teach more complex skills that comprise, in part, the simpler skills taught earlier. The LisaGuide system itself caused users problems, in addition to those caused by the training strategy adopted by the system, these problems are detailed in Carroll and Mazur (1986) and Carroll (1990), and will not be described further here.

4.5.3 Basic user interaction

Direct manipulation systems, of which the Lisa is an example, are assumed to be easier to learn than command-based user interfaces, the learning costs of remembering commands and of understanding the effects of commands are presumably reduced. Direct manipulation systems, however, do not remove the need to learn the simple operations and commands that must be understood by the user. Basic user operations involving the mouse must be learned before more complex tasks may be attempted. Examples of the simple operations that make up human-computer dialogues with direct manipulation systems are termed clicking, pressing, selecting, and dragging. Even this terminology proved confusing to users, explanations of these terms were not provided in the Lisa's documentation, yet more complex tasks were described in terms of these operations. Although these skills were taught and practised using the LisaGuide, they were not named, users were forced to make the (hopefully correct) association between their action and the effect on the system, and then to relate their action to a concept named in the documentation. Carroll (1990) reports that this problem can trouble users even after over an hour of using the system.

Double clicking of a mouse button to perform operations was also a cause of users' problems. Acceptable delays between the first and second click proved difficult to judge, and some users were never able to open and run applications by methods involving a double mouse click. One user was reported as hypothesising the effect of a double mouse button click, but he attempted to confirm this hypothesis using a file which did not respond to a double click and was confused by the resulting unexpected system response. Where clicking was applied to on-screen objects, again the lack of an explicit association between a skill and its name, or between the effect of an action and its name, caused users problems. One user was unable to perform tasks that required a particular icon to be selected and become *highlighted* until he

deduced that when an icon *darkened* in response to a mouse click it was in fact highlighted.

4.5.4 The Desktop

According to Carroll (1990: 62), with the Apple Lisa:

> "The user is encouraged to think of the display as a desktop containing objects that can be manipulated on analogy with physical manipulation. This approach attempts to make learning a computer easier by designing interface actions, procedures, and concepts to exploit specific prior knowledge that users have of other domains. Instead of making the interface simpler, this approach seeks to increase the initial familiarity of actions, procedures and concepts that are already known."

Above, problems caused by the use of a typewriter analogy to explain text-editing systems were described. These problems also arose in the Lisa system, space characters were inserted into documents when the user expected these characters to replace and overwrite unwanted characters in the document.

As with basic user interaction, the vocabulary used to describe objects on the electronic desktop also caused users problems. Users seemed unable to associate the terms "clipboard", "stationery pad", "typing", "tear-off stationery" and "folders" with the analogous on-screen objects presented, data structures or tasks. Where objects *were* understood, when users attempted to apply skills from the real world in the electronic domain, they discovered that these skills were not supported. "Tearing off" a sheet of stationery from the on-screen pad of paper used to create short notes and documents proved difficult to perform. One user was seen making sweeping

motions with the mouse, clicking the mouse button while the pointer was over a corner of the pad mimicking the action performed in the real world. Objects in the electronic domain were also found to not support the steps making up more complex tasks in the order in which the steps would be performed in the real world.

Some of the basic concepts underlying the desktop metaphor were criticised by Carroll and Mazur. On-screen objects are divided into data files (which resemble documents, pictures and folders), functions (for example file copying is denoted by a photocopier), and application tools (such as word processors and spreadsheet software). Carroll and Mazur (1986: 41-42) describe the notion of an application tool as:

> "... a good example of an ancillary metaphor too general to imply anything useful."

Data files are the product of application programs, but users preferred to perform tasks directly using the data file, rather than open the application that produced it and to view the data file as the data manipulated by the application. It also seems difficult to reconcile application tools with the desktop metaphor, it may be asked what meaning dragging an application onto the desktop means, applications having no immediately obvious real-world analogue.

4.6 Discussion: Metaphor and System Learning and Use

Metaphors are often proposed as a design solution to the problem of creating usable computer systems. This chapter discussed models of metaphorical and analogical reasoning and understanding in the learning of previously unfamiliar computer systems. Users can obtain the information needed to use an unfamiliar computer system from a number of sources. One source is the documentation supplied with a

system, in the form of manuals, on-line tutorials, and instruction on audio- or videocassette (for example Apple, 1983). It is well known that users resist using manuals whenever possible (Carroll and Rosson, 1987), and even when manuals are used they may not best support users. The Lisa study described above again shows users' reluctance to use manuals, as does the Macintosh study described below. The manual supplied with the Lisa system is an example of a manual that fails to support users in their efforts to learn the system. The learning of a computer system has been likened to being immersed in a foreign culture — a major requirement of learning the new system is to learn the language used to describe it. Carroll and Mazur (1986) found that many simple skills required to use the system were described using terms that were not described in the documentation. Users also often change the wording of tasks into their own familiar vocabulary, as will be demonstrated in the study of novice Macintosh users described in the next chapter. Mayer (1981) found that allowing users to put instruction material in their own words increased the time taken to learn a domain, but increased the quality of their learning. Manual authors seem not to recognise this, and tasks become more difficult to learn and perform as a result. This *vocabulary problem* (Furnas et al., 1987) is well-recognised by others. Even when many aliases and synonyms for commands or objects in many task domains, are provided, studies show that the probability of the system designer and users using the same word when attempting to name the same action or concept tends to be very low .

The learning mechanism assumed by the Lisa and Macintosh manuals seems to be that described by Anderson (1982). In Anderson's model skills are acquired by the compilation over time of a declarative representation of a problem, or task to be performed, into productions which are used in the performance of routine cognitive skills, the initial declarative representation eventually being lost. This model accounts for how people are often able to perform tasks (knowing how) without being able to articulate the process by which a task is performed (knowing what).

More complex skills and tasks are described in terms of being built up from simpler existing skills. In this way, the Macintosh documentation teaches menu use by first teaching the simpler skills of manipulating the on-screen pointer using the mouse and clicking the mouse button at appropriate times. This model of skill acquisition is realised in the ACT* and ACT-R models of cognition (Anderson, 1983, 1993). Criticisms, however, of the ACT* model as applied in human-computer interaction are presented in Lansdale and Ormerod (1994).

Waern (1990) suggests that two aspects of learning need to be considered in human-computer interaction, what the user already knows and what the user has to learn. Accounts of learning assume that in order to acquire new facts and skills, the learner must already possess considerable knowledge. Another source of information that can aid a user to learn a new computer system is transfer of existing skills and knowledge to the new domain. If systems support or encourage the use and transfer of existing knowledge, Waern (1985) proposes that this may not aid learning the new system if *negative transfer* occurs and existing knowledge interferes with the actual knowledge required to use the system. Singley and Anderson (1989) conclude from their studies of learners using a different text-editing system from a familiar one that transfer of existing skills can only take place at the level of individual productions, the smallest unit of cognitive skill. Transfer from one direct manipulation interface to another can be achieved if the interfaces are sufficiently similar. This can be seen in (Young, 1981) and in the study of Macintosh users described above where users could be seen trying to load data files into an application by typing MS-DOS commands into a text entry field in dialogue boxes rather than selecting files from a list shown.

As direct transfer of existing knowledge cannot account for all the knowledge required to use an unfamiliar computing system, analogical processes become more important in learning and using a computer system. Many of the problems of specific

user interfaces, systems, and computing domains have been described above. In general, the process of analogical reasoning in humans presents difficulties for user interface software designers and users. In the cases considered above, users are provided with an analogical source domain, either in the form of the model world represented on-screen, or in some form of instruction. Studies of humans provided with a useful analogy before attempting a problem solving exercise shows that the analogy may often not be applied to solve the problem (Gick and Holyoak, 1980, 1983). Some evidence of this is seen in the Macintosh study presented above where some subjects seem incapable of applying knowledge about waste baskets (which are depicted on-screen) in order to plan and execute methods of retrieving files from the waste basket or trashcan.

Where direct transfer of knowledge cannot be achieved, Singley and Anderson (1989) suggest that a more declarative representation of a domain or problem must be retained in order to facilitate analogical processes in reasoning. Singley and Anderson state, however, that if the target domain differs from what is expected from the result of analogical mapping, the problem solver is unable to resolve and explain mismatches. In the Copycat model of analogical reasoning, where no mapping can be made due to there being no concept in the target domain that satisfies an analogical mapping from the source concept, the problem server is faced with the problem of *conceptual slippage* (Mitchell, 1993). If no mapping can be made from a source concept, the set of possible sources must be relaxed, or must slip, until a concept that can be mapped is found. Conceptual slippage is realised in the Copycat model in a *slipnet* knowledge representation structure.

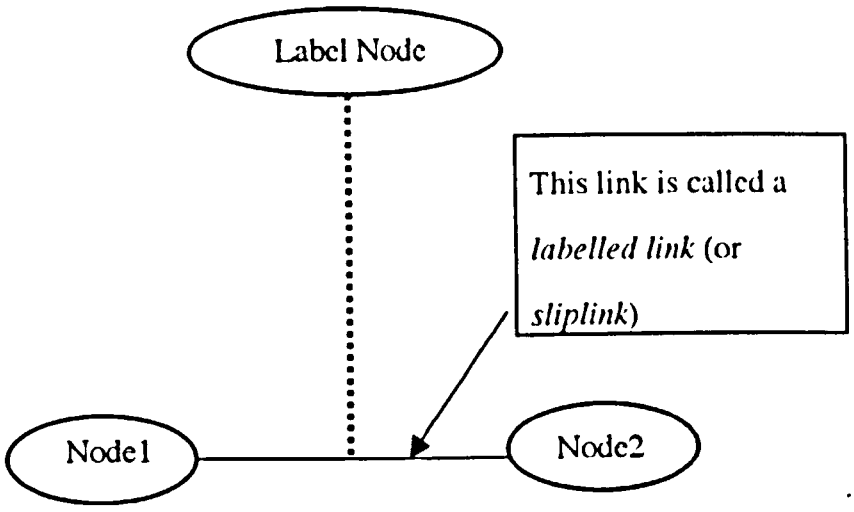


Figure 4.4 Building blocks of slipnets (French, 1995: 57).

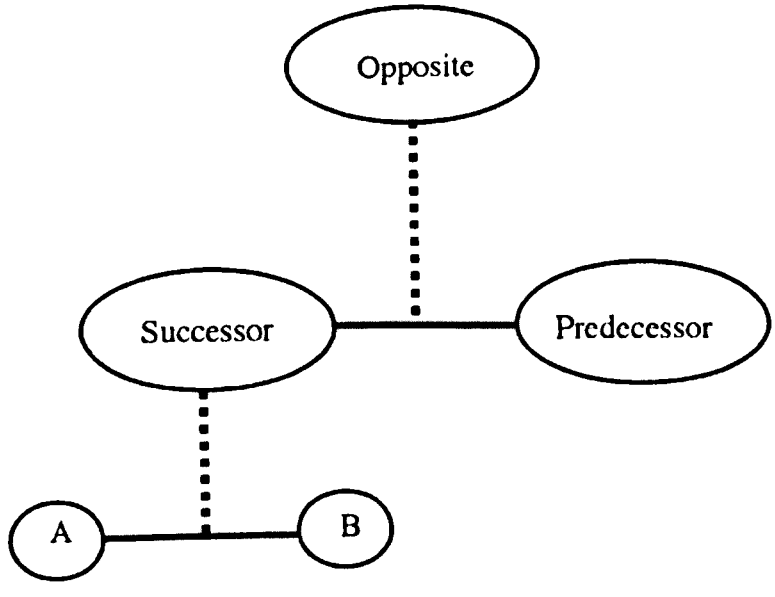


Figure 4.5 Part of a slipnet representing the alphabet (French, 1995: 58).

In a slipnet (Figures 4.4 and 4.5), concepts are represented as nodes. Activation of nodes can spread to concept nodes that map less well to suitable concepts in the target domain. This multivalued model of concepts that may possibly be mapped

analogically explains why in Copycat there is no *one* single mapping made every time the mapping is made. Instead,, a number of different problem solvers will choose different source concepts. The possible patterns of activation may result in a plausible (according to the slipnet, and indeed the problem context) analogical mapping being made, but which is unwanted, unhelpful, or unintended. If, as Smith (1987) suggests, all user interfaces will present behaviour and features that cannot be explained in terms of the metaphor represented, the means by which mismatches are explained, and the mechanisms used to generate accounts of mismatches require investigation. Mismatches between an analogical source and a target computing system are considered by the active learning approach to systems.

An important issue raised by analogies being made by stochastic processes such as in Copycat, and TableTop (French, 1995), is that there is no one "correct" analogical mapping, there are only mappings that are more likely to be made. Some inferences may have such a high probability of being chosen that their selection can be almost assured, but the mechanisms underlying such models cannot rule out an unlikely inference being chosen by surviving with a sufficiently high excitation level. Indeed, some analogical inferences made can be described as "almost sick" (Hofstadter, 1985: 575). The nature of the conceptual models captured in slipnets means that the knowledge possessed by the system making the analogy confuses and complicates the process, and the making of analogies between domains will always be subjective and imprecise.

Analogies are employed to ease the learning and use of a computing system by encouraging the use of existing skills and knowledge in the new domain of the computing system. It is possible, however, for a graphical user interface to adopt an analogy, but one which gives users no suggestion as to the skills that may be applied. The user interface shown in Figure 4.6 attempts to realise a graphical user interface which adopts the analogy of, and which is intended to be as flexible in the uses to

which it can be put as, a sheet of paper. However, as Buxton (1993) has stated, no suggestion as to how to use the interface is given to the novice user. Dialogue with this system is initially very *under-determined* (Thimbleby, 1980).

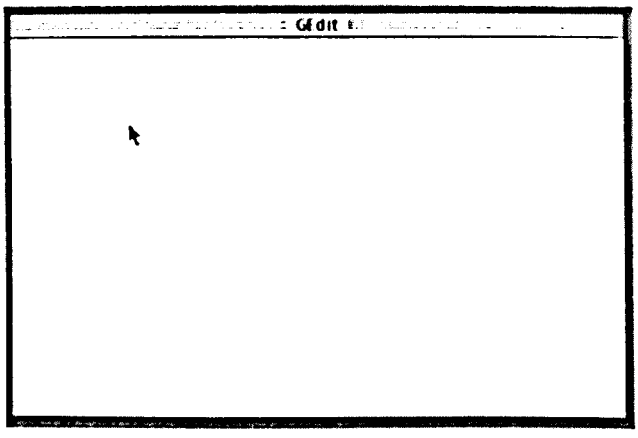


Figure 4.6 GEdit, a paper-like interface[3].

4.7 Types and Theories of Metaphor

Mentioning the ideas of Douglas Hosftadter and his colleagues in the Fluid Analogies Research Group introduces the major division in the types of theories of metaphors that have been previously presented. While metaphor has been a topic of thought and debate since the time of Aristotle, theories of metaphor have yet to achieve sufficient influence in the understanding of metaphor itself, let alone in the understanding of cognition in general, for it to be possible either to undertake an historical survey of theories from either Karl Popper's or Thomas Kuhn's viewpoints of scientific progress. Instead, theories of metaphor can merely be said to fall into one of two categories, *comparison theories*, or *interaction theories*.

---

[3] Developed by Gordon Kurtenbach (Kurtenbach and Buxton, 1991), all interaction with the system is by means of gestures communicated to the system via the mouse.

Comparison theories, which account for what Indurkhya (1992) terms *similarity-based* metaphors, rely on there being some underlying similarities between the (familiar) source and the (unfamiliar) target domains to permit meaning to be transferred. We can see that comparison theories are assumed by those, such as Halasz and Moran (1982), who denounce the use of metaphor as a way of explaining interactive devices and who state that metaphor plays little part in explanation and that it can only serve as a rhetorical flourish. According to Ortony (1993) Aristotle held similar views. The majority of theories of metaphor that have been applied to date in HCI and user interface design are comparison theories despite interaction theories of metaphor predating the first user interface metaphors.

4.7.1 Interaction Theories

A number of interaction theories of metaphor have been proposed — Indurkhya (1992) surveys those that were the most well-formulated at the time he was writing. Below we shall describe and make use of the interaction theory due to George Lakoff and his colleagues. We shall follow Indurkhya's (1992) claims that Lakoff's theory is an interaction theory even though Lakoff himself, according to Indurkhya, sides his theory more with comparison theories. The principles of an interaction theory of metaphor were stated best by Max Black (1993: 27-28)[4]:

> "1. A metaphorical statement has two distinct subjects, to be identified as the 'primary' subject' and the 'secondary' one...
> 2. The secondary subject is to be regarded as a system rather than an individual thing.

[4] Black's paper was originally written for the first edition of the volume in which it appears which was published in 1979.

3. The metaphorical utterance works by 'projecting upon' the primary subject a set of 'associated implication' comprised in the implicative complex, that are predictable of the secondary subject...

4. The maker of a metaphorical statement selects, emphasizes, supresses, and organises features of the primary subject by applying to it statements isomorphic with the members of the secondary subject's implicative complex...

5. In the context of a particular metaphorical statement, the two subjects 'interact' in the following ways: (a) the presence of the primary subject incites the hearer to select some of the secondary subject's properties; and (b) invites him to construct a parallel implication-complex that can fit the primary subject; and (c) reciprocally induces parallel changes in the secondary subject."

A possible explanation for why Lakoff chooses, according to Indurkhya (1992), to speak of his theory of metaphor as a comparison theory is because he and his colleagues (for example Lakoff and Johnson, 1980) often analyse metaphors that are called "dead" by some authors (for example, Mac Cormac, 1985). These are utterances that are metaphors but which no longer possess any power to shock or surprise because they are such everyday aspects of speech. Where Lakoff's theory can be terms an interaction theory is in regarding the things being metaphorically compared as rich *systems* of relationships (the "implicational complex"), rather than single objects. While structure-based approaches to domain representation are systems of relationships, they have until recently lacked the complexity and richness that Lakoff's theory has addressed from the beginning. Forbus (2001) signals a shift by one group of structure-based analogy researchers to consider more complex knowledge structures.

Interaction theories are important in HCI for two other reasons. The first, which will considered as further work, is the way in which interaction between domains gives rise not just to understanding of the unfamiliar domain, but it also causes changes to the understanding of the familiar domain. Interaction theories do not just address common-place, or dead, metaphors, but seek to explain *novel* metaphors, or what Indurkhya (1992) terms *similarity-creating* metaphors. How our user interface metaphors change our *existing* conceptual structures and perhaps subsequently constrain the ways in which we can imagine interaction with computing devices is a topic that remains for further work. The second reason why interaction theories are important, which HCI has not considered in great depth to date, but which is central to the Lakoff theory, are the issues arising in system use, cognition and understanding from the fact that users are *embodied.* Indurkhya (1992: 402) makes the following observations of the interaction theory he presents, which like the systems developed by Hofstadter, French, Mitchell and other members of the FARG, particularly considers physical actions performed by the 'user': "The model would work by producing a conceptualization of the target sensorimotor data set in terms of the source concepts. The resulting representation would be metaphorical, if it would be something that the system would not have produced by itself when the source were not explicitly given." Interaction theories stress that action and models of how actions are performed must be considered as part of source domains if target domains are to be metaphorically understood. The Lakoff theory of metaphor described below is valuable for directly addressing this concern.

4.7.2 Metaphor and Analogy

In user interface design, as mentioned, the words "metaphor" and "analogy" appear to others to be synonymous. It may be asked if this is simply another example of the limited understanding of metaphor that HCI as a whole possesses, if the mental processes that underlie these tropes are the same, or if these "tropes", as exhibited by

user interfaces, are the same. However, the use of terminology in the metaphor and cognitive science literature makes these questions difficult to answer currently.

Donald Schön's (1983) analysis of professional's behaviour in terms of them being *reflective practitioners* views discovery and hypothesis generation as part of this form of practice as new problems being framed in terms of descriptions that are perceived to be similar to previous experience. This framing is termed *seeing-as* by Schön. For Schön, analogy is synonymous to a form of metaphor termed *generative metaphor* which is when seeing-as occurs when the domains of experience are different. Schön's key example of this, which is often cited elsewhere, is of a group of designers who were trying to replicate the properties of paintbrushes with bristles made of natural materials with bristles made of artificial materials. Copying superficial properties of bristles did not give useful results, but when one of the designers that Schön was studying observed that "...a paintbrush is a kind of pump!" (Schön, 1983: 184) the designers were able to replicate the key aspects of brushes and to consider new designs. For Schön, only generative metaphors create insight in this way and allow people to not only understand the unfamiliar domain, but also to develop a new understanding of the familiar, presumed fully understood, domain.

The nature of the similarity between domains may influence whether the process of understanding one domain in terms of another is metaphorical or analogical. A particular problem that affects some user interface designs, as will be discussed further in the following chapter, is that the model world does not behave in a causal way. Even if the software does not betray the user's notions of causality, as Spiro et al. (1989: 507) note "Some analogies are very effective at characterising surface features and relationships but gloss over underlying causal mechanisms. The result is that learners tend either to fill in a convenient but incorrect causal account of their own, or just leave the causal mechanism unexplained as a kind of 'black box.'" They go on to suggest that "It might be said that a comparison based primarily on surface

descriptive aspects is more metaphorical than analogical. However, our point here is that an underlying relational structure is indeed transferred — that is, people have a tendency to interpret metaphors analogically." (op. cit.: 507).

Fraser (1993: 332) holds that this connection is even stronger than Spiro et al. believe. He defines a metaphor as "an instance of the non-literal use of language in which the intended propositional content must be determined by the construction of an analogy." Fraser is clear to point out that he does not regard metaphor and analogy as synonymous, but that analogy is a process that must be involved in the understanding of a metaphor. Gentner and Jeziorski (1993) view metaphor as a broad category encompassing analogy, matches that map structure independently of object descriptions, and other kinds of matches between domains. Gentner's work has been primarily concerned with analogy understanding, in which the structure-mapping process is the primary mechanism. Many of the figurative expressions studied by Lakoff and Johnson (1980) are claimed by Gentner and Jeziorski to be analogies rather than metaphors. Gentner et al. (2001) attempt to unify metaphor with analogy, more precisely to claim an equality between the set of metaphors and the set of analogies, and hence to be able to account for metaphors by their structure-mapping model of understanding analogies, but do not completely succeed. Some forms of metaphor cannot be described in terms of their unified account, although, like Schön, they state that novel metaphors can be accounted for by processes of analogical understanding. In the more problematic case of what they term *conventional metaphors*, where the base term refers both to a literal concept and a metaphoric category, metaphors are said to have a *career*. The career of a metaphor can be likened to the change in fuzzy category membership in Mac Cormac's model, metaphors that are at first novel can, in Gentner's et al. view be handled in structural terms. As the metaphor becomes more familiar, understanding it is more the task of determining its category membership, no further change in understanding of either source or target domain is possible, and it becomes what Mac Cormac would call

dead. It at this point requires no structure-mapping processes to understand when encountered again.

In the Lakoff/Johnson theory, the relationship is reversed. Rather than (some, most, or all) metaphors being comprehensible in terms of processes and mechanisms of analogical reasoning, analogies are forms of metaphor. Lakoff, from analyses undertaken by Mark Turner, suggests that the general mechanism of analogical reasoning is the GENERIC IS SPECIFIC metaphor. This metaphor maps schemata onto their generic-level schemata. The following sections define the term schemata and present the details of the Lakoff/Johnson theory of metaphor. Among the advantages provided by this theory as a tool for understanding user interface metaphors, is the provision, through the existence of generic-level schemata, of ways of addressing the fundamental questions of how novice users make a mapping between a known domain (the metaphorical source) and a completely unknown domain, and how they can formulate motor sequences to interact with this domain.

4.8 Is Metaphorical Understanding of User Interfaces Possible?

Above, the problems of existing metaphor-based user interfaces have been discussed, as well as the difficulties of attempting to develop new metaphors for model world interfaces to computing systems from an understanding of previous work in understanding metaphorical and analogical reasoning. In this section we consider a serious objection to the world view underlying much of the work discussed above. This objection is one that, while it suggests that metaphors and analogies should have no place in the design of graphical user interfaces, we will take as the starting point to employ a particular recent interaction theory of metaphor understanding as another tool to use to criticise existing metaphor-based user

interfaces and to employ in the analysis and design of two new interface designs that will be presented later in the thesis.

The work that has had the greatest influence on current work on metaphor and analogy in user interface design adopts the *Objectivist* world view. This model of reality, meaning and reference, long-standing in Western science and philosphy, adopts the following assumptions:

"- Thought is the mechanical manipulation of abstract symbols.
- The mind is an abstract machine, manipulating symbols essentially in the way a computer does, that it, by algebraic computation.
- Symbols (e.g. words and mental representations) get their meaning via correspondences with things in the external world. All meaning is of this character.
- Symbols that correspond to the external world are *internal* representations of *external* reality.
- Abstract symbols may stand in correspondence to things in the world independent of the peculiar properties of any organisms.
- Since the human mind makes use of internal representations of external reality, the mind is a *mirror of nature*, and correct reason mirrors the logic of the external world.
- It is thus incidental to the nature of meaningful concepts and reason that human beings have the bodies they have and function in their environment in the way they do. Human bodies may play a role in *choosing* which concepts and which modes of transcendental reason human beings actually employ, but they play no essential role in *characterising* what constitutes a concept and what consitutes reason.

- Thought is *abstract* and *disembodied*, since it is independent of any limitations of the human body, the human perceptual system, and the human nervous system.
- Machines that do no more than mechanically manipulate symbols that correspond to things in the world are capable of meaningful thought and reason.
- Thought is atomistic, in that it can broken down into simple 'building blocks' - the symbols used in thought - which are combined into complexes and manipulated by rule.
- Thought is logical in the narrow technical sense used by philosophical logicians; that is, it can be modelled accurately by systems of the sort used in mathematical logic. These are abstract symbol systems defined by general principles of symbol manipulation and mechanisms for interpreting such symbols in terms of 'models of the world'." (Lakoff, 1987: xii-xiii)

This model of the world is one that has become subject to recent severe criticisms, these criticisms will need to be taken into account as further work in role of metaphor and analogy in human-computer interaction is undertaken. The severest criticism of the Objectivist world view has been provided by Hilary Putnam (1981) who has provided a well-known theorem that refutes many of the assumptions of Objectivism listed above. Putnam's Theorem is stated as follows (Putnam, 1981: Appendix):

"Let *L* be a language with predicates F_1, F_2, ..., F_k (not necessarily monadic). Let *I* be an interpretation in the sense of assigning an intension to every predicate of *L*. Then if *I* is non-trivial in the sense that at least one predicate has an extension which is neither empty nor universal in that at least one possible world, there exists a second

interpretation *J* which disagrees with *I* but which makes the same sentences true in every possible world as *I* does."

Putnam uses this theorem to show that a sentence such as "a cat is on a mat" can be transformed so that "cat" refers to "cherry" and "mat" refers to "tree" without changing the truth value of the sentence in any possible world. Predicates of this form make up the models of the target and source domains between which an analogical mapping is made in the structure mapping and ACME models. Putnam continues by stating that "a more complicated reinterpretation...can be carried out for all sentences of a whole language. It follows that there are always infinitely many different interpretations of the predicates of a language which assign the 'correct' truth-values to the sentences in all possible worlds, *no matter how these 'correct' truth-values are singled out.*" (Putnam, 1981: 35, original italics).

The effect of Putnam's Theorem is to render meaningless the notion of *reference*, the connection between symbols in the mind and distinct objects in the external world that can be categorized. While the impact of Putnam's Theorem has been felt most fully in the fields of cognitive science, philosophy and linguistics, its results also apply in the prospects for theories of metaphorical understanding. Lakoff (1987: 172) states that:

"The Objectivist paradigm also induces what is known as the *literal-figurative* distinction. A literal meaning is one that is capable of fitting reality, that is, of being objectively true or false. Figurative expressions are defined as those that do not have meanings that can directly fit the world in this way. If metaphors and metonyms have any meaning at all, they must have some other, related literal meaning. Thus, metaphor and metonymy are not subjects for objectivist semantics at all. The only viable alternative is to view

them as part of pragmatics - the study of a speaker's meaning. Moreover, it follows from the objectivist definitions of *definition* itself that metaphor and metonymy cannot be part of definitions. They cannot even be a part of concepts, since concepts must involve a *direct* correspondence to entities and categories in the real world (or a possible world)."

Putnam (1981: 72-74), in addition, says that :

"Even if the notion of a 'similarity' between our concepts and what they refer to doesn't work, couldn't there be some kind of abstract isomorphism, or, if not literally an isomorphism, some kind of abstract *mapping* of concepts onto things in the (mind-independent) world? Couldn't truth be defined in terms of such an isomorphism or mapping?

The trouble with this suggestion is not that correspondences between words or concepts and other entities don't exist, but that too *many* correspondences exist. To pick out just *one* correspondence between words or mental signs and mind-independent things we would have already to have referential access to the mind-independent things...This simply states...the intuitive fact that to single out a correspondence between two domains one needs some independent access to both domains."

This independent access to domains between which a mapping is to be drawn, this God's eye view of the world as Putnam terms it, is what Putnam's Theorem states can never be available. Therefore much of the work on metaphorical comprehension and analogical reasoning, assumed to a considerable extent in human-computer

interaction, becomes subject to Mitchell and Hofstadter's (1995: 290) criticisms of these systems. They claim that there is nothing in the existing models of analogy making that makes "symbols stand for anything in a recognisable way. Only the person who used them to encode 'pieces of knowledge' sees them as standing for anything." The conclusion drawn, therefore, is that metaphor will never allow users to understand the systems they use, if concepts and models of the system are to be formed to a large part by metaphorical mappings between domains modelled in a set-theoretic, Objectivist, way. It might be concluded, as a result, that users should be provided only with literal accounts of the system they are to learn and use. In doing so, however, devising training material would prove difficult and we would reject the advantages that some forms of mental models give to users. We would also be rejecting the pervasive nature of metaphor and analogy in understanding the world (Lakoff & Johnson, 1980). Before presenting the design of the Medusa system, therefore, we are required to explore to some extent the role that metaphor will play in its design and the theory of metaphor that is assumed in its model world, in doing so, we continue a programme begun by Rohrer (1995)[5].

4.9 Cognitive Semantics of User Interface Metaphors

Below, the motivations behind a new interface design named Medusa are discussed. The design of this user interface, while trying to avoid the problems posed by existing systems based on metaphors and analogies discussed above, appreciates that analogy and metaphor are an inescapable part of learning and understanding. In recent years, the Objectivist world view underlying the understanding of metaphorical and analogical reasoning assumed in the theories of metaphor

---

[5] Very recently, use of the Lakoff/Johnson theory of metaphor understanding in HCI has also been adopted by others, for example Benyon and Imaz (1999), but these authors have yet to consider the number of interface designs and task domains considered in this thesis.

described earlier in this chapter has come under criticism. These criticisms have led to the development of a contemporary theory of metaphor by George Lakoff and his colleagues (Lakoff, 1993). Below, an attempt to formulate this theory of metaphor in terms of user interface design is presented. This formulation of the theory is employed to underlie one version of the Medusa user interface design that will be presented in Chapter 6.

4.9.1 Image Schemata and Metaphorical Projection for Understanding

Putnam's Theorem gives rise to the conclusion that metaphor can play no part in our understanding of concepts within an Objectivist world view. Mappings between domains, as assumed in existing work on metaphorical understanding of user interface model worlds, teach us little about the quality of a user interface metaphor according to Putnam's Theorem as the concepts and representations of the external world modelled have no connection to mind-independent things. We must therefore confront how meaning is obtained and what role metaphor can play in the world. The conclusion drawn by Lakoff (1987) and Johnson (1987) is that meaning is grounded in terms of *image schemata*, and that the world can be understood in terms of these schemata and metaphorical mappings from these schemata to describe a situation or statement. Johnson (1987: 28-29) provides the following definition of the term "image schemata":

> "On the one hand, they are not Objectivist propositions that specify abstract relations between symbols and objective reality. There might be conditions of satisfaction for schemata of a special sort (for which we would need a new account), but not in the sense required for traditional treatments of propositions. On the other hand, they do not have the specificity of rich images or mental pictures. They operate at

one level of generality and abstraction above concrete rich images. A schema consists of a small number of parts, images, and events. In sum, image schemata operate at a level of mental organisation that falls between abstract propositional structures, on the one side, and particular concrete images, on the other.

The view I am proposing is this: in order for us to have meaningful, connected experiences that we can comprehend and reason about, there must be pattern and order to our actions, perceptions, and conceptions. *A schema is a recurrent pattern, shape, and regularity in, or of, these ongoing ordering activities.* These patterns emerge as meaningful structures for us chiefly at the level of our bodily movements through space, our manipulation of objects, and our perceptual interactions." (Original italics)

We shall give one example of understanding based on image schemata before considering a number of case studies applying the Lakoff/Johnson model of metaphor comprehension to aspects of existing user interface designs. Johnson (1987: 32) presents examples of how a small number of image schemata based on experience of IN-OUT relationships can account for many uses of the word "out" in English. Figures 4.7, 4.8, and 4.9 show these uses with depictions of the schema[6] being relied upon for understanding.

[6] These schemata were devised for Claudia Brugman's 1981 University of California at Berkeley MA thesis *The Story of Over.*

John went out of the room.

Pump out the air.

Let out your anger.

Pick out the best theory.

Drown out the music.

Harry weasled out of the contract.

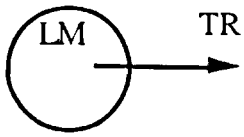


Figure 4.7 The OUT_1 schema

Pour out the beans.

Roll out the red carpet.

Send out the troops.

Hand out the information.

Write out your ideas.

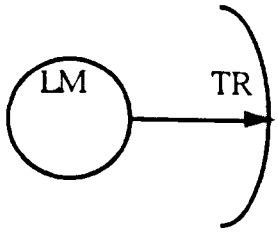


Figure 4.8 The OUT_2 schema

The train started out for Chicago.

Figure 4.9 The OUT_3 schema

It should be noted that these depictions of image schemata are only *depictions*, but they serve to illustrate the bodily experiences captured in, and described by, the schemata. In the schemata shown LM is the "landmark" in relation to which TR the "trajector" moves. Considering the schema OUT_1 and the sentence "John went out of the room", the circle (LM) represents the room as a container, and John moves along the arrow (as TR) out of the room. The diagram does not represent information, such as the shape of the room (which may not be circular), or the vector along which John moves, but it instead "gives only one idealised image of the actual schema...It is, rather, a continuous, active, dynamic recurring structure of experiences of similar spatial movements of a certain kind." (Johnson, 1987: 36).

The intention behind the Lakoff/Johnson theory of metaphor is to be able to account for human understanding of concepts and language. In order to do so it must be able to describe the non-physical as well as the physical. To this end, all understanding is achieved by metaphorical extensions from image schemata. An example given by Johnson (1987: 35) is the sentence "I don't want to leave any relevant data out of my argument." This relies upon the OUT schema in order to be understood, but also relies upon the ARGUMENT IS A CONTAINER metaphor, claimed to be a very common metaphor in Western culture (Lakoff and Johnson, 1980).

There are, obviously, more schemata in addition to IN-OUT used to ground our comprehension of the world. Johnson (1987: 126) presents a partial list of schemata (shown in Figure 4.10), which he claims are "*...pervasive, well-defined, and full of sufficient internal structure to constrain our understanding and reasoning.*" (original italics). Some of these schemata will be used below in case studies, and in the following chapter where the details of a revised design of the Medusa system are presented, to be useful in attempts to provide meanings of and to discuss interaction with aspects of graphical user interfaces.

CONTAINER	BALANCE	COMPULSION
BLOCKAGE	COUNTERFORCE	RESTRAINT REMOVAL
ENABLEMENT	ATTRACTION	MASS-COUNT
PATH	LINK	CENTER-PERIPHERY
CYCLE	NEAR-FAR	SCALE
PART-WHOLE	MERGING	SPLITTING
FULL-EMPTY	MATCHING	SUPERIMPOSITION
ITERATION	CONTACT	PROCESS
SURFACE	OBJECT	COLLECTION

Figure 4.10 Some pervasive image schemata

4.9.2 The Lakoff/Johnson Theory in HCI

A criticism that can be made of the use of metaphor in HCI is that few authors who apply a particular theory of metaphor comprehension, or who apply particular theories of metaphor comprehension in their interface designs, state which theory they are adopting. The Lakoff/Johnson theory is different in that most authors that employ it explicitly state that they are adopting, or testing, this theory. However the entire body of literature on use of the Lakoff/Johnson theory in HCI prior to (Treglown 1999; 2000) comprised (Rohrer, 1995). The use of the Lakoff/Johnson theory in later parts of this thesis was begun independently of Rohrer's work, and we have considered a larger number of case studies (reported in Chapter 6) than he did. This should not be taken as a criticism of Rohrer's efforts, he merely returned to his work in cognitive science and linguistics after providing his contribution to HCI.

Rohrer's (1995) contribution is to show that some aspects of the Apple Macintosh user interface can be analysed and reasoned about in terms of the Lakoff/Johnson theory of metaphor. Much of (Rohrer, 1995) addresses the trash can which was

found in the emprical study reported on in Chapter 3, and in work reviewed in this chapter, to be particularly problematic. We shall return to this user interface feature, and Rohrer's examination of it, in one of the case studies undertaken in Section 5.x. Rohrer's interest in metaphor comes from his claiming to "... see not only a tension between literal and magical qualities of metaphor, but a tension between the users' feeling that the computer is an extension of their bodies and believing that it is an 'other' - a sentient being witha consciousness of its own." We shall return in Section 5.4.3 to the teaching strategies employed by Rohrer to explain the trash can, and other features of the Macintosh user interface, including explaining it in terms of another agent with which to communicate. It is in terms of Rohrer's interest in ideas of embodiment and their impact on theories of cognition and interaction, and his idea that "The magical features of the DESKTOP metaphor are inextricably bound up with the users [sic] aesthetic feel of the system" that the Lakoff/Johnson theory deserves to be considered as a candidate theory of metaphor in HCI and provides explanatory power that other theories of metaphor do not yet offer to HCI in the level of detail at which they are currently described.

An example that Rohrer gives where embodiment plays a key role in understanding, or appreciating, an interface design feature is the use of "zooming" of windows as they open. A closed window can take many forms, in early WIMP systems, as was mentioned above, they were denoted as an icon on the root window. In the Microsoft Windows 95 user interface, a closed window is denoted by a rectangular region, or button, on the task bar along the bottom of the screen. Clicking on a button on the task bar will open or close the associated window depending on its current state. In most systems (including the Macintosh, when an icon is double-clicked on) the window opens by expanding, or zooming, from a small region of the screen to occupy a far larger region. Rohrer describes this enlargement as "zooming", mimicking the enlargement of a document as ones head moves toward it, or as it is brought closer to the viewer. To Rohrer "Zooming is more than just a nice touch... it

is one of the best examples of how user interface design can draw on common patterns of feelings. Zooming is a pattern of feelings that takes place *in and through time*; the realisation that all feeling takes place in and through time is the most important step in thinking about users' bodies. Part of being embodied is being a creature in time, and being in time is part of what the Cartesian theory of mind and ideas as objective entities hides from our attention." The closest to an account of zooming provided in the cognitivist tradition in HCI is Barnard and May's (1995) configuration of Barnard's ICS (Interacting Cognitive Subsystems) cognitive architecture to comprehend transitions between displays and depictions of system and program states that are borrowed from the grammar of film-making, and accounted for by theories of film. Instead of borrowing methods from the visual arts, and trying to explain how the mind can comprehend them in order to answer the questions "Why make windows zoom? Why make an event that could happen instantaneously on the computer screen take longer than necessary simply for the sake of metaphorical consistency?" For Rohrer "Making users feel at home in their user interface is important to the development of users [sic] abilities to imagine and intuit how the user interface will work." Zooming, to Rohrer, is an example of an image schema that captures a familiar pattern of experience with the physical world, one that Kosslyn (1994) claims can be rehearsed and visualised by the human capacity for mental imagery. Beyond mere explanation of how certain computer animations can be accounted for, Rohrer argues that "... the aesthetics of user interface design requires thinking about subjective, preverbal bodily patterns of feeling... The development of good user interfaces depends on careful phenomonological and psychological research on subjective bodily experiences." These are what HCI has typically, so far, ignored. These aspects of interface will be returned to at the end of the next chapter, and at other later points in the thesis.

4.10 Conclusions

This chapter has considered the use of analogies and metaphors in user interface design. Although analogies are suggested as a solution to the problem of designing usable software, as the review above discusses:

- Analogies are often the source of difficulties for learners;
- Human ability to make use of analogies provided to aid problem solving is limited;
- Where mismatches occur between the source and the target computer system the mechanisms by which mismatches are explained are poorly understood;
- Analogy-based user interfaces may not suggest ways in which they may be used, and existing analogy-based systems can prove difficult to learn and use when performing realistic tasks in realistic setting.

Within HCI metaphor has typically been seen as only one mechanism for, or description of, understanding of computing systems. The following chapter surveys some of these other mechanisms and places metaphor in a wider context of these models and mechanisms.

Chapter 5

Users' Models of Interactive Systems

"The situation gets quite confused, with people turning different knobs, the effects of which they have no way of knowing."

— John Cage.

5.1 Introduction

In Chapter 4 the drawbacks of adopting the use of analogies and metaphors in the design of interactive user interfaces were examined. In particular, the recognition phase (finding a suitable source domain) and the elaboration (mapping between domains) phase of the analogical reasoning process identified by Hall (1989) were considered. Where the mapping gives rise to mismatches, and elements of the source domain cannot be applied in the target domain, the model of the target domain employed must be evaluated and debugged. The final phase of analogical reasoning identified by Hall is to consolidate repairs of the target domain model in order to improve future performance.

The debugging of target domains has been addressed in models presented by Burstein (1986) and Adelson (1989). Both of these models rely on understanding causal mechanisms in the source and target domains in order to remove inferences from the target model that cannot be applied and used, and to add mechanisms within the target model in order to account for the behaviour of the target model.

This process depends on the target domain being capable of being described in causal terms, some interface designs, however, have been implemented which do not display causal behaviour. In an attempt to examine direct manipulation user interfaces based on analogies and metaphors, efforts were made to formalise knowledge of the physical world and to examine user interfaces based on physical world analogies. Considering an earlier version of the Macintosh Finder, it was noted that the behaviour captured in the action-effect rule (Monk and Dix, 1987) shown in Table 5.1 is exhibited. In Treglown (1994) it was shown possible to model moving files around the Macintosh file system and deleting files from the file system using the notation of Qualitative Process Theory (Forbus, 1984). It, however, proves impossible to describe the behaviour in Table 5.1 in terms of the objects visible on-screen and physically realistic changes to attributes of the objects (such as spatial position) alone. Norman (1983) notes that some users' behaviour when using interactive systems involves superstitious beliefs. In this case, without generating an account of the system behaviour in terms of mechanisms in the underlying software, the system cannot be described in terms of any understanding of physics, either existing or conjectured (Sheldrake, 1994).

R1: <Drag file icon over the disk icon and release mouse button>:: File icon appears in window associated with disk if window is open and file is copied across. The trashcan empties if 'full' and the trashcan icon shows 'empty'.

Table 5.1 An action-effect rule describing partial behaviour of the Macintosh Finder interface.

The conclusion derived and assumed in the remainder of this thesis is that the use of analogies and metaphors in user interface design as currently assumed does not provide an ideal solution to the problem of designing user interfaces. This conclusion is not original, it is one arrived at by Halasz and Moran (1982), among others. They

argue that metaphor should be used as they claim it is used in literature, in passing, and to convey meaning at a particular point in the development of understanding, not in an attempt to describe the entire system. Even where a number of analogies are used to describe different aspects of the system, users may apply the incorrect analogy to describe the part of the system currently of interest (Rumelhart and Norman, 1981). In order to relate a number of less all-encompassing, more specific metaphors, Halasz and Moran (1982: 384) conclude and suggest that systems be explained in terms of a *conceptual model*:

> "... to present the underlying conceptual structures directly to the user, providing him with an appropriate basis for reasoning about the system."

Norman (1998: 180-181) also discourages the use of metaphors, saying:

> "Basically, those who espouse the use of metaphors are giving human-centered development a bad name, almost as bad as those who believe in 'user-friendly' systems... Designers of the world: Forget the term 'metaphor.' Go right to the heart of the problem. Make a clean, clear, understandable conceptual model."

A similar conclusion is reached by Laurel (1993). She suggests that what have been termed user interface metaphors, an assumption employed in the previous three chapters, are in fact user interface *similes*, which assert that one concept is *like* another. Metaphor-based systems are intended to resemble the metaphorical source, but may differ. Smith (1987) suggests they will always differ. Differences between the source and target domains and the problems they pose for users were the subject of the previous chapter. User interface similes act as a mediator between three concepts, the real-world object, the representation of the object, and the functionality

and data structures implemented by the computing system denoted by the representation. The user is forced to form "mental models of what is going on *inside the computer* that incorporate an understanding of all three parts" (Laurel, 1993: 130).

In the remainder of this chapter, the models of computing systems formed and employed by users in order to understand metaphor-based systems and perform tasks using such systems, are discussed. This discussion will form the basis for presenting user interface designs for supporting tasks supported by the sorts of metaphor-based systems discussed in Chapter 2. These new designs attempt to overcome some of the drawbacks found with metaphor-based systems and to present a useful conceptual model to users.

5.2 Types of Users' Models of Systems

Users are said to have knowledge of two aspects of the systems they use, *task knowledge* ('how to do it' knowledge), and *device knowledge* ('how it works' knowledge). Carroll and Olson (1988) list three types of model to account for these two aspects:

- knowledge of simple sequences of commands and key-presses, learned by rote and memorised with little or no understanding,
- methods (more complex task knowledge), and
- mental models.

Task knowledge is often described in terms of Card, Moran, and Newell's (1983) GOMS (Goals, Operators, Methods, and Selection Rules) model, extensions of GOMS and alternative models and notations that express similar knowledge. Problems arising from use of GOMS in human-computer interaction are well known.

GOMS addresses only routine cognitive behaviour of the sort eventually realised by the ACT* model of learning and supported by traditional documentation design, and it addresses errors in human performance only if knowledge used to overcome errors can also be described in terms of GOMS. For novice users of a new system any routine skills that might be applied in the new system are those that can be transferred from their knowledge of existing systems.

Users' device knowledge is thought to be represented in the form of their mental models. The term mental model has given rise to much confusion, being applied to a number of different entities, so much so that the term *mental muddles* has been used in some discussions. Norman (1983) offers some additional terminology in order to resolve misunderstandings, he suggests that four notions need to be identified in this discussion, the *target system*, the *conceptual model* of the target system, the user's *mental model* of the target system, and the *scientist's conceptualisation* of the mental model. The conceptual model is the creation of a system designer or teacher, it is intended to provide "an appropriate representation of the target system, appropriate in the sense of being accurate, consistent, and complete" (Norman, 1983: 7). Through interaction with the system, people are said to construct mental models (although the psychological support and evidence for mental models is still the subject of debate). Mental models are structures that evolve with time and with use of the system and are formed to represent the conceptual model as represented by the *system image*, the view of the conceptual model represented on-screen and in system documentation. These notions do not identify all the models that may be held by the agents identified by Norman. The designer's model of the system is the conceptual model, the user's model of the system is their mental model, the researcher will possess the scientist's conceptualisation of the mental model (a model of a model). It is also possible, as seen in adaptive systems and intelligent tutoring systems, for the system to possess a model of the user. The conceptual model and the scientist's conceptualisation of the mental model will be focused on below.

Norman (1983: 8) makes the following observations on the nature of mental models which need to be considered if system design taking into account mental models is to be undertaken:

> "1. Mental models are incomplete.
> 2. People's abilities to 'run' their models are severely limited.
> 3. Mental models are unstable: People forget the details of the system they are using, especially when those details (for the whole system) have not been used for some period.
> 4. Mental models do not have firm boundaries: similar devices and operations get confused with one another.
> 5. Mental models are 'unscientific': People maintain 'superstitious' behaviour patterns even when they know they are unneeded because they cost little in physical effort and save mental effort.
> 6. Mental models are parsimonious: Often people do extra physical operations rather than the mental planning that would allow them to avoid those actions; they are willing to trade-off extra physical action for reduced mental complexity. This is especially true where the extra actions allow one simplified rule to apply to a variety of devices, thus minimizing the chances for confusions."

Carroll and Olson (1988) discuss four types of mental models: *metaphors*, *networks*, *glass box models* and *surrogates*. The notion of surrogate models is due to Young (1983) who also introduces a number of device models: *strong analogies*, *mappings* (or task/action mappings), *coherence*, *vocabulary*, *problem spaces*, *psychological grammars* and *commonality*. Metaphors, and hence strong analogies, have been discussed in detail above, the remaining models mentioned by Young (1983) which are not discussed in detail here, have received little further attention. The exception

is the use of problem space models (Card, Moran, and Newell, 1983) which Young (1983) initially ignored since problem spaces are mostly used to model of routine cognitive skill of the sort described by GOMS, where problem solving when using devices is ignored. Young's views have since altered (Young and Simon, 1987; Simon and Young, 1988) to where problem spaces are thought to form the basis of an approach to interaction in which planning and routine cognitive skill are opposite ends of the same continuum. Where skills to perform tasks and subtasks do not exist, plans that make use of what appropriate skills the agent does have can be constructed using well-known backwards-chaining methods into which existing skills can be interleaved. This work also gives rise to an important claim that only a limited amount of planning will be undertaken in HCI tasks. Where planning is needed, it is closely tied to execution of planned actions, and can only be partial as much planning is often a matter of the user revising their intentions in response to system feedback following previously performed actions.

5.2.1 Networks

Generalised transition networks (GTN's) have been proposed as a model of interactive systems (Keiras and Polson, 1983). GTN's comprise nodes representing states, and arcs that represent transitions from one state to another and the actions performed on registers in response to user input. The complexity of graphical notations for finite state machine models of systems is partially overcome in the GTN by permitting sub-networks to be described separately and called recursively from nodes in other networks. GTN's are employed as device models in Keiras and Polson's Cognitive Complexity Theory (Keiras and Polson, 1985), in which task knowledge is represented in a form of a GOMS model. GTN's are also proposed as a means of determining how easy the system is to learn and remember. This is achieved by comparing a user's GTN model of the system with the actual, ideal, GTN. Missing nodes and arcs in the user's model denote missing or faulty

knowledge. The types of systems that may be described easily by GTN's are limited, however. GTN's are a difficult model in which to model interleaving (Cockton, 1992), which must be supported in order to realise systems such as Rooms and task switching in other systems such as the desktop. GTN's, as described by Keiras and Polson (1983), adopt a stimulus-response view of systems, the systems described produce feedback and change state in response to obvious commands, such as control characters and typed commands. In the direct manipulation systems described in Chapter 2, commands may consist of hundreds of discrete events that are interpreted as a high-level user intention. Although a GTN may be used to realise such systems in user interface management systems, it seems unlikely that the user's model of the device includes awareness of, or makes use of, the events generated by devices such as a mouse. The sense of engagement with the mouse that users are intended to develop with accurate tracking of the on-screen pointer as the mouse is moved precludes such detail being made available to the user. GTN's, as described by Keiras and Polson (1983), describe systems that consist of a single state altered in response to user input. In the design and implementation of object-based systems, such as the desktop and the Alternate Reality Kit, the system's state is distributed over a number of on-screen or software objects. This is another manifestation of the problem of interleaving which is not, as discussed above, addressed well by the GTN either in terms of system implementation or users' models of the system.

5.2.2 Glass Box Models

Glass box models (Du Boulay, O'Shea, and Monk, 1981) are said to combine elements of both metaphors and surrogate models (which are described below). Glass box models are intended to provide a perfect mimic of the target system, but provide some semantic interpretation, in the form of metaphors, of relevant components. Glass boxes are based on an analogy of cut-away windows in physical

devices to reveal some of the workings inside. This notion forms part of the idea underlying a glass box model, that interaction with components should be achieved with simple input and simple forms of output, and that hidden actions and states should be *illuminated.* Glass box models are more prescriptive than descriptive, concepts are introduced in order to describe the notional machine in order to perform some task. Carroll and Olson (1988) relate glass box models to operational metaphors; rather than attempt to provide novices with an account of existing programming languages, however, the glass box concept has been employed in the design of novel programming languages and simplified interfaces to existing programming languages. Being prescriptive models, however, the glass box offers little in the way of description of the sorts of models of systems that users may generate and use.

5.2.3 Surrogates

A surrogate model is based on the notion of a "working model", an account of a system intended to explain how a system works. A surrogate should perfectly mimic the target system's input and output, but the process by which the system's output is produced need not be the same as realised by the target system, and the internal workings of the surrogate need not be isomorphic to the target system. Young (1981, 1983) attempts to construct surrogate models of some simple computing devices, namely three types of pocket calculator:

- a simple four arithmetic operation calculator,
- a calculator supporting more complex algebraic expressions, and
- a calculator based on the reverse polish notation which relies on a stack as a store of numbers input and postfix use of arithmetic operators.

While Young was able to provide surrogate models of the four function calculator and the reverse polish notation calculator, the other calculator, which accepts more complex algebraic expressions, was shown to be more difficult to construct a surrogate model of. This is because no account could be provided of how the system responds to 'ungrammatical' sequences of key presses.

In addition to the difficulty of providing an account of some aspects of the calculators in terms of registers, stacks, and how data is placed in — and moved between — registers and the display, Young (1983) doubts the psychological validity of surrogates. He claims that they are of little use for describing routine behaviour (although they have a greater role in problem solving and predicting the outcome of novel calculations), and that the cognitive workload of employing a surrogate model may be too large for users. The calculators studied, even if they could produce results to ill-structured sequences of key presses, rely on well-formed sequences of key presses in order to produce meaningful results. The structure of commands, key presses and other user input actions accepted by devices and required in order to perform tasks is the concern of another form of knowledge about systems, *task/action mappings*. These are multi-levelled models developed from Moran's (1981) Command Language Grammar which describe the interface between the user and an interactive system, and the transformation of a user's task to the sequence of actions needed to perform the task on the system. The models of the calculators that Young describes are referred to as 'implied register models' (Young, 1981), they introduce registers, internal state and other data structures sufficient to describe the outputs produced by the calculators. In general, however, Carroll and Olson (1988: 51) observe that:

> "... while the surrogate always provides the right answer (the one the target system would have generated) it offers no means of illuminating the real underlying causal basis for the answer. It is a

good, complete analogy that may allow the user to construct appropriate behaviour in a novel situation, but does not help the user why the system behaves the way it does."

System Learning using Surrogate Models

Young (1981, 1983) presented surrogate models of a number of pocket calculators, but did not explore how providing learners with these surrogates might affect learning and use of the calculators. Halasz and Moran (1983) considered learners' use of a reverse polish notation calculator when groups of learners were provided either with no surrogate, or a surrogate model of the calculator. The surrogate model taught was intended to provide a problem space in which learners could invent operator sequences and carry out problem solving. This problem space consisted of the calculator's stack and rules describing the changes in the stack's contents as the user pressed keys. Halasz and Moran (1983) found that the learners who were not provided with the model performed routine tasks up to 40% faster than the learners who were provided with the model. Young (1983) suggests that surrogates play no role in routine behaviour, the results of Halasz and Moran suggest that surrogates impose an extra load on the user when performing routine tasks. When performing *invention tasks* where operator sequences must be planned in order to perform novel calculations, learners provided with the model were able to perform tasks 15% faster than the group of learners who were not provided with a model.

Keiras and Bovair (1984) employed a novel device in their examination of the role of surrogates in learning. They also adopted the method of comparing users who were taught only rote procedures for using the device with those users who were taught an explanatory model of the system in addition to the rote instructions. This explanatory model was in the form of a "cover story" and a diagram describing the topology of the internal components of the device. Providing a model to the learners

meant that the model group learned the procedures needed to use the device faster, and learned more efficient procedures. The model group also retained knowledge of procedures needed to use the device better than the group taught only the rote procedures, and remembered more efficient procedures. In a second experiment, Keiras and Bovair (1984) examined the effect of the device model provided on learners' inference of procedures performed when using the device. They found that learners provided with the model took fewer actions to infer procedures than the other group, and, once they had learned a first procedure, took fewer actions to infer a second procedure to perform the same task. A third experiment examined which of the "cover story" and the description of the topology of the device's components is the important factor in learning.

The third experiment performed by Keiras and Bovair (1984) suggested that the topology of the device's components is the important factor in a device model presented to learners. Keiras (1992) presents further experiments that confirm this suggestion. The topology of the device components is an aid to learning even in systems where the internal state of device components is made apparent to the user. Kieras (1992) is able to derive a set of guidelines for the design of device models, especially for diagrammatic displays presented on-screen. These guidelines are listed as follows (Keiras, 1992: 893-894):

> *Topological structure*
>
> Show the topological and causal structure of the system, such as the pathways between components, controls and indicators using conventions that are visually clear. Structural relationships involved in understanding system states must appear on the diagram.

Control and indicator states

Echo the topological effects of external controls, and show indicator states at the corresponding topological points on the system diagram.

Internal states

If information on the states of internal components is reliable and available, show the states that are significant to the user, so that there are no hidden states and no inferences are required to deduce significant component states. Provide the state information at the corresponding topological point in the display.

Causal relationships

Show the pathway of causality through the topological structure, such as the colour-coding of energized connections. Distinguish component states from other state information that may be on the displays (for example, by using different colour-codes).

Malfunctions

Show failures of causal flow, such as malfunctions, in a perceptually salient way (for example, bright yellow for a component that fails to produce output when it should).

If one considers Carroll and Olson's (1988) comment about the surrogate model's lack of ability to describe causality and possible reasons for some system behaviour and if one also considers Keiras and Polson's design suggestions, it is clear that further attention should be paid to the roles that device component topology and causal relationships between components play in mental models. .

5.2.4 Task-action Mappings

Above an action-effect rule for the Apple Macintosh user interface was given. Such rules capture the ways that the display and relevant aspects of the system state change in response to actions performed by the user. Task-action mappings are also rules, but instead capture the user's tasks and the actions that they must perform in order to achieve these tasks. The following generic rules (taken from Schiele and Green, 1990: 60) show the deletion of a character of text in a number of Macintosh-based application packages in the notation of the Task-Action Grammar:

R2 "Delete a single character"

```
T [Unit=char, extent=1, Effect = remove, Clipboard=no]:=
      MOUSE-point(%location) + MOUSE-click +
      edit [Unit=char, Effect=remove, Clipboard=no]
```

```
R9    edit [Unit=char/word/cell-entry/object,
            Effect=remove, Clipboard=no] := "BKSP"
```

Task-action mapping models, while proving useful in judging the consistency and learnability of commands (Lee et al., 1994; Howes and Young, 1991), and while having strong claims to being psychologically meaningful (Schiele and Green, 1990), suffer from considerable drawbacks as a means of modelling interactive systems in their basic forms. Basic task-action mapping models ignore the role of the display in system use, it is assumed that a sequence of actions will perform the task irrespective of display contents. The D-TAG and E-TAG versions of the Task-Action Grammar were developed to overcome this limitation. Another drawback of task-action mappings lies in their application to metaphor in interface use. The model presented by Rieman et al. (1994) implements in the ACT-R and SOAR cognitive architectures task-action mapping rules for a direct manipulation user interface and also implements a process model of metaphorical system use similar to

Hall's (1989). The resulting systems transform the knowledge of running an application by double clicking on its icon by an analogical process to derive the task to run another application. While valuable in concentrating on a task and how task knowledge may be metaphorically extended, the model that Rieman et al. (1994) present is very limited and, for example, provides no account of how metaphorical understanding might have led to the initial method for running an application being acquired.

5.2.5 Qualitative Models as Mental Models

Keiras's (1992) design suggestions present an account of device models that matches Olson's (1992) definition of a mental model:

> "Mental models are knowledge that the user has about how something works, its component parts, the processes, their interrelations, and how one component influences another."

A similar definition is provided in Halasz and Moran (1983). Given this definition of mental models, a potentially productive method of modelling and exploring mental models, supported by existing notations and software tools, might be to employ qualitative reasoning (Bobrow, 1985). The use of artificial intelligence methods in the study of mental models has been previously suggested (Decortis et al., 1991), and the use of qualitative reasoning in an HCI domain is observed by Payne (1991a). Qualitative models have to date been employed in the modelling of physical systems (Gentner and Stevens, 1983; Hayes, 1985) and of simple devices or simple components within complex systems (Bobrow, 1985). Owen (1986) suggests that the study of human-computer interaction would benefit from investigating qualitative (or "naive") models of computing systems, but he does not provide any such accounts himself. Payne (1991a), however, does explore users' understanding and models of a

computer-based system derived from their existing knowledge and inferred from the system's behaviour.

The qualitative notations and algebras that have been devised and employed to date to model systems and devices other than computer-based systems are of interest as they may be used to confront the fact that computing hardware is not infinitely fast (Dix, 1987). On-screen objects may be subject to uncertain delays in rendering and screen updates, and functionality provided by the underlying software may engage in lengthy computations, or may make remote procedure calls where network delays become noticeable. Qualitative notations which are able to model change over time and rates of change may prove of increasing interest in the description of mental models. In (Treglown, 1994), an attempt was made to use Forbus's (1984) Qualitative Process Theory (QPT) notation to model device knowledge to account for mismatches between a metaphor-based graphical user interface and the actual behaviour of on-screen objects. This work is summarised below by discussing models of the behaviour of on-screen objects in the Alternate Reality Kit system, the mismatch between describing the task/action mapping and user feedback of moving files around a desk top metaphor system's file space, and the underlying functionality.

In the QPT notation, systems are described in terms of objects, which have a number of attributes, and processes that act on objects to alter their attributes. It was shown possible in (Treglown, 1994) to describe the attributes of on-screen objects in user interfaces based on a physical world metaphor using QPT. A number of attributes, denoted by the Quantity-type predicate, that may be associated with data files, and the attributes associated with a text file are shown in Figure 5.1. *Processes* can act on objects to change attributes of objects when a number of pre-conditions hold. These pre-conditions depend upon factors that lie outside the QPT model, the amount of water in a bath cannot increase until someone turns on a tap, for example. Conditions

that can be described by the QPT model, termed Quantity-conditions, also determine whether a process is active or not, and whether it is able to affect those attributes of the object that the process acts upon. While processes are active, *influences* (other values relevant to the model) may alter values of attributes directly, or may have an effect of qualitative proportionality where the relationship and influence are less well defined.

```
Quantity-Type(size)
Quantity-Type(creator-application)
Quantity-Type(size-if-run)
Quantity-Type(printable-object)

doc a document
Has-Quantity(doc, size)
Has-Quantity(doc, creator-application)
Has-Quantity(doc, printable-object)

video a video-fragment
Has-Quantity(video, size)
Has-Quantity(video, creator-application)
```

Figure 5.1 QPT notation attributes and on-screen objects

This approach to modelling systems, in terms of objects, attributes, and processes is termed *object-centred.* This approach was adopted in (Treglown, 1994) over *device-centred* approaches to qualitative modelling (de Kleer & Brown, 1984; Kuipers, 1985) as the process-centered models make minimal reference to hidden values and mechanism within the system which determine the system's behaviour. Methods to determine and elicit mental models are still early in their history as topics of research

(Payne, 1991; Rutherford and Wilson, 1992; Carroll and Olson, 1988; Rogers, 1992). Attempts such as Payne's (1991) exploration of the mental models formed of simpler computing devices have not yet been undertaken with more complex systems such as those described in Chapter 2. Therefore, it is uncertain how much of the system's mechanism is apparent to users, and can be modelled in a device-centered notation. The *User Virtual Machine* notation (Tauber, 1988) has been employed to describe the mental model of a software application. It can model the state of hidden components of which users are aware and which they employ when performing certain tasks, but it does not model components which may be perceived or inferred by users. The notation also makes no reference to the temporal behaviour of changes of state of the components modelled.

Smith (1987) observes that his Alternate Reality Kit displays behaviour that cannot be termed either literal to the metaphor or magical, this behaviour cannot be described in terms of the physical world metaphor on which the system is based. An example he gives is the increasing delays in updating the display as the number of on-screen objects in an ARK simulation, and hence the system load required to compute and render the display increases. This causes moving on-screen objects to move in an increasingly 'jerky' and unrealistic way. Figure 5.2 shows a proposed model for a freely moving object in an ARK simulation where objects are not subject to friction or any gravitational or frictional forces. This model is able to describe the behaviour of such an object, but must include the influence on the speed of the object of the computational load on the system.

```
Process Motion(B, dir)

Individuals:
   B an object, Mobile(B)
   dir a direction

Preconditions:
   Free-Direction(B,dir)
   Direction-Of(dir, velocity(B))

QuantityConditions:
   Am[velocity(B)] > ZERO

Relations:
      A[velocity(B)] ∝Q- system-load
Influences:
   I+(position(B), A[velocity(B)])
```

Figure 5.2 A QPT model of a moving object in an ARK simulation

In desktop interfaces, the task of moving a file is often performed by the user pressing the mouse button while the pointer is over the icon denoting the file, by dragging the icon until it is over a particular folder, and by finally releasing the mouse button. If one assumes that a metaphor-based system supports direct engagement, and that users perceive the on-screen object to actually *be* the data file of interest to them (Hutchins, Hollan, and Norman, 1986), it should be possible to model the movement of an on-screen object, and hence of the file it denotes within the file space, in a process such as that shown in Figure 5.3. This process is more complex than one needed to model on-screen objects in many metaphor-based systems, objects tend not to have a perceivable mass, although computer-based simulations that employ input devices with force-feedback can communicate a sense of an object's mass to the user. The Aristotelian idea of motion described in this model, where objects require that a constant force be applied to them in order for them to move at a constant rate reflects dragging an on-screen object.

```
Process motion

Individuals:
     B an object, Mobile(B)
     dir a direction

Preconditions:
     Free-Direction(B, dir)
     Direction-Of(dir, net-force(B))

QuantityConditions:
     Am[net-force(B)] > ZERO

Relations:
     let velocity be a quantity
     velocity ∝Q+ net-force(B)
     velocity ∝Q- mass(B)

Influences:
     I+(position(B), A[velocity])
```

Figure 5.3 A QPT model of motion (Forbus, 1984: 134).

The feedback in the display should be immediate and appropriate as the on-screen pointer and file icon being dragged track the user's movement of the mouse. The computation performed when moving the data making up a file from one directory in the file space to another directory, however, has a temporal duration. It is possible to model this computation in QPT as shown in Figure 5.4.

```
Process move-file

Individuals:
   source-file an object,
      Has-Quantity(source-file, size)
   destination-file an object
   src a folder
   dest a folder,
      Has-Quantity(dest, free-space)
   path a data-path,
      Connection(data-path,src,dest)

Preconditions:
   (T task-is-move-file)
   Aligned(path)

QuantityConditions:
   A[free-space(dest)] > A[size(source-file)]
   A[size(source-file)] > ZERO

Relations:
   Let move-rate be a quantity
   A[move-rate] > ZERO
   move-rate ∝Q+ device-speed(dest)
   move-rate ∝Q- system-load

Influences:
   I- (size(source-file), A[move-rate])
   I+ (size(destination-file), A[move-rate])
```

Figure 5.4 A QPT model of moving a file within the underlying system functionality

Despite the action performed in the on-screen model world being intended to be analogical to the computation performed by the underlying application, these two proposed mental models represent different physical processes. These modelsare difficult to map between using the structure-mapping model of analogy and the framework of learning physical domains provided by Forbus and Gentner (1986). The temporal duration of the process performed by the functionality of the application implies that direct engagement with the file in the file space breaks down for this task and metaphor.

5.3 The Role of the Display as Source of Information in System Learning and Use

Another aspect of design that needs to be considered is the role that the display plays as a source and store of information about the system state, both when learning a system and in routine use of a system. Lansdale and Ormerod (1994) class the knowledge and skills needed to learn and use human-computer interface software into three types:

1. skills as procedures,
2. skills as understanding, and
3. skills as exploration.

Skills in procedural form are explored and described in models such as GOMS and ACT* which tend to rely on what is termed the traditional "systems" approach to manual design and instruction. The problems with the systems approach to instruction have long been documented, as have the resulting problems in developing the routine cognitive skills modelled using GOMS and its variants.

Viewing skills as understanding explores the forms of mental models of systems and their role in learning and system use. As have been examined above, some forms of mental models make skills in the form of understanding difficult to attain. While some forms of mental models have a useful role in system use, others, like metaphors, which attempt to aid initial steps in system learning and use have been shown to often create more problems for users than they may solve. Some models, like transition networks and metaphors, also prove incapable of representing every aspect of the model world and underlying functionality of the system they are intended to provide a useful representation of.

Understanding user skills as exploration requires examining the way in which users extract information from the interface and the way user interfaces may be designed to minimise the amount of features of the interface that users are required to learn. Such skills are examined by the study of *display-based reasoning*. Mayes et al. (1988) observed that experts' recall of command names on the pull-down menus of a screen-based word processor did not differ from that of novice users of the system. This result contrasts with many results in cognitive psychology where experts are found to have greater recall of meaningful patterns in the task domain than novices and people with intermediate levels of expertise. Mayes' et al. observation suggests that the interface itself is being employed as a form of externalised memory. Operators may be observed in, and inferred from, the display, and the effect of operators on the system state tend to be observed rather than learned (Payne, 1991b). Hence, rather than learn the system, users simply *use* it. This view of user interface design and use requires some consideration be paid to the use of metaphor, analogy and prior knowledge and issues in problem solving, affordances and planning in learning to use a previously unfamiliar computing system.

There are difficulties with reliance on users to use the interface itself as a store of information and a form of externalised memory. Users may not explore the system fully and potentially useful functionality may not be discovered. Users may not even explore the system *sufficiently* to discover the functionality that supports their tasks (Lansdale and Ormerod, 1994). Another problem caused by reliance on display-based reasoning is that users may be forced to perform repetitive low-level actions to achieve some tasks. Frolich (1993) notes an increase in the importance of the converstation notion in human-computer interaction over the model-world notion in the mixed mode systems of which most metaphor-based systems are examples. Repetitive tasks and tasks which would require a reasonable length of time to perform may be delegated to a virtual partner or *agent* (Cypher, 1990). Such agents

can be hard to reconcile with the metaphors chosen for many user interface designs, such as those explored in Chapter 2.

5.4 Using the Lakoff/Johnson Model for Analysis and Design of User Interfaces

We have introduced the Lakoff/Johnson model of metaphor, in which experience and sentences in a natural language either appeal directly to image schemata, or can be understood by a metaphorical mapping to image schemata. It is our claim that the Lakoff/Johnson theory of metaphor allows us to analyse and critique existing user interface designs, and provides a method for judging the success of new user interface designs. Below, the Lakoff/Johnson theory is applied to novel or problematic aspects of existing user interface designs in order to demonstrate its explanatory ability. In using the Lakoff/Johnson theory in this way a claim stronger than the suggestion that metaphor *is part of* a way of understanding on interactive system (or mental model) is being made. While we do not claim (as Lakoff and Johnson do not) that *all* understanding is metaphorical, Indurkhya (1992) surveys the theories of a number of authors that do make this claim, the Lakoff/Johnson theory is a theory of semantics. It should therefore be complete and sufficient to account for users' understanding of a device, or to give reasons why a problematic device has poor usability. In order to test that the Lakoff/Johnson theory can explain user interfaces that are problematic, or which are hard for other HCI models to explain, a number of case studies are presented below. As mentioned in the previous chapter, Rohrer's (1995) work is described in more detail in Section 5.4.3 while the notorious trash can is examined. This follows two more case studies that consider systems that Rohrer (1995) did not address.

5.4.1 Case Study 1: An Immersive Environment

The artist Char Davies' virtual reality installation work "Osmose" (Davis, 1996; Wertheim, 1999), shown in Figure 5.5, presents the challenge of understanding how users obtain enough meaning from it in order to successfully interact with it. The Osmose model world consists of a number of levels, depicted in Figure 5.6, containing a forest of semi-transparent stylised trees, free floating words taken from texts by post-modernist authors and the Osmose source code. Users have no representation of the hand, or other cursor, they can only move within the Osmose world and between levels, but they are passive in being able to see objects, but not handle them.

Figure 5.5 A view inside Osmose (Wertheim, 1999: 39)

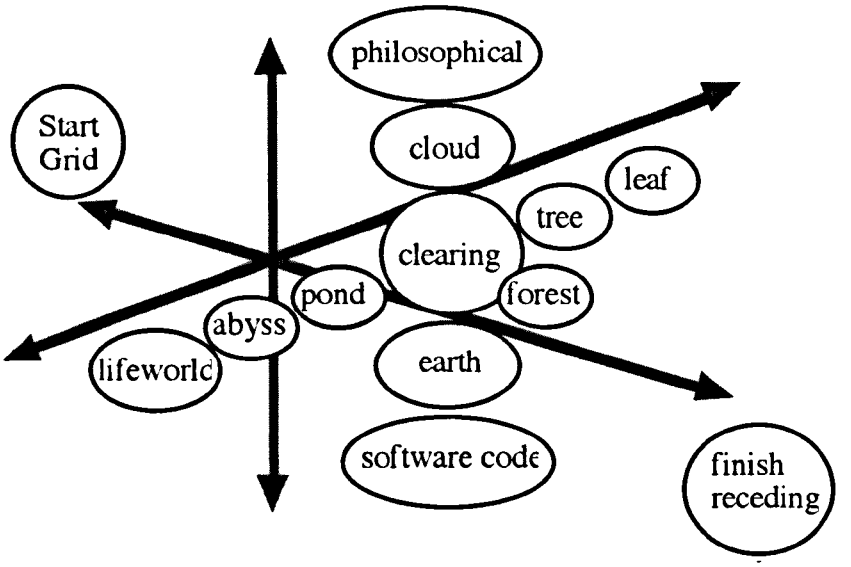


Figure 5.6 The Structure of the Osmose model world

Input to the computing machinery that runs Osmose comes from 3D position trackers on a headset and vest worn by the user, and from a strap placed around the user's chest which measures respiration, in the same way that polygraph or lie-detector machines do. In fact the strap is taken from just such a device. In order to move in the horizontal plane, users must lean in the direction that they wish to travel, they then drift in that direction within Osmose until they return to an upright standing position. In order to move between layers, the user must adjust their breathing and fill their lungs to float up, or empty their lungs to sink. Such a novel environment, with its use of novel input devices, is difficult to model, but it cannot be impossible for users to devise and attempt suitable actions and movements of the body in order to move within the world, otherwise the Osmose world would go unused.

Considering the user's motion between layers of the Osmose world, interaction would appear to rely on the COUNTERFORCE image schema, depicted in Figure 5.7.

Figure 5.7 The COUNTERFORCE schema

Motion up or down depends on the amount of air in the user's lungs, as is motion when diving, the experience that inspired the Osmose system's development. The 'upward' force on the user derives from Archimedes' principle and is proportional to the 'buoyancy' of the user. The 'downward' force on the user derives from the user's 'weight'. By controlling their breathing, the user is able to adjust the balance of forces and can float up or sink down through the medium in which the user is suspended. This aspect of interaction requires very little metaphorical extension to the COUNTERFORCE schema, all that is required is a mapping from the air that surrounds the user to the medium that gives the user buoyancy in the Osmose world, a mapping such as the metaphor AIR IS A FLUID. By its use of an additional novel input device, Osmose is able to provide functionality to support "flying" in an immersive environment that is as natural to the user's physical experience as possible. Other virtual realities (Weimer and Ganapathy, 1989; Fisher et al., 1986) provide far less satisfactory approaches. In these systems in order to request a menu that appears before them as a free-floating panel (from which they must select an option) the user must make a special grasping gesture. Subsequently the hand-shaped cursor acts not as a grasping facility, but as a positioning facility for the user's point of view. By adopting this approach, direct manipulation is replaced in this mode by the conversation paradigm of user interaction, an unseen agent (the computing hardware) must be informed as to the way in which subsequent user input from a dataglove is to be interpreted.

While interaction with the Osmose system based on the "floating" experience can be accounted for by the Lakoff/Johnson theory, movement in the horizontal plane must also be explained. The BALANCE schema; "consisting of force vectors (which can represent weight as a special case) and some point or axis or plane in relation to

which those forces are distributed. In every case, balance involves a symmetrical (or proportional) arrangement around a point or axis" (Johnson, 1987: 85); suggests a common experience which may form users' understanding of this aspect of interaction with the system. In leaning in the intended direction of travel, equilibrium is disturbed and a net force is created in that direction[1]. The user will then move, Osmose having Newtonian physical laws in its model world, until an arresting force is created as the user regains an upright position. It seems, therefore, that the BALANCE schema can be appealed to almost directly in order to understand this aspect of interaction with Osmose.

5.4.2 Case Study 2: Snap-Dragging

Drawing and computer-aided design application software is often required to support the precise placement of line segments and other shapes. Various facilities can be provided for this, including the displaying of grid points within the drawing area of the application's window onto which objects may be accurately placed, and the use of constraint systems. These mechanisms are limited, however, as some drawing tasks can be difficult to perform, and that some relationships between drawn objects and line segments may be difficult to maintain if one object is moved. Bier and Stone (1986) present snap-dragging as a better alternative to grids and constraints. Presentation of Bier and Stone's system can be found in their article, our task is only to consider how it can be understood and used by the system's user, and to explore the possible role of the Lakoff/Johnson theory of metaphor in the process of understanding snap-dragging.

---

[1] We see this interaction style implemented in the Segway personal urban transportation device which has two wheels, one either side of the rider, unlike other "scooters" where wheels are aligned one in front of the rider, one behind. Computers in Segway cause it to steer and accelerate depending on the way in which the rider leans their body.

The snap-dragging system differs from other drawing packages in that the cursor does not hold and move the drawing implement or tool directly, but is used to pick up a "caret" which is subject to attractive forces generated by the cursor and other objects such as line segments. As the cursor moves, the caret moves with it unless it becomes attracted to an artifact, such as a circle of particular radius or line extending from an existing line segment, that was defined by the user beforehand using simple commands on a pop-up menu. The artefact appears for a short time as the caret approaches it. Unless the user moves the caret away from the artefact, it will snap-drag to the point or line. Other shapes may then be drawn precisely from a point of intersection or tangent. The principle schema which can appealed to for understanding in Bier and Stone's system is ATTRACTION, depicted in Figure 5.8. Once again, this schema represents a pervasive physical experience, and the snap-dragging system, in its use of animation when tracking the cursor position requires little metaphorical extension in order to be understood. Johnson (1987: 38), though, allows attracting vectors to be either actual or potential and allows for the existence of additional objects in order to describe a situation. In a computer-aided design drawing in which there are a number of snap-drag artefacts, users might observe chaotic behaviour in the movement of the caret.

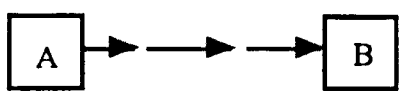


Figure 5.8 The ATTRACTION schema

In Bier and Stone's snap-drag system, the ATTRACTION schema may be appealed to directly for understanding as the source of the attractive force is visible. Other systems, however, exhibit snap-dragging, or snap-to-a-grid behaviour, but are more complex to understand. Some implementations of the DESKTOP metaphor, for example, do not allow icons on the desktop or workbench, or within open folders, to

be placed at just *any* position. While icons may be placed on top of another, obscuring or hiding the one below, the number of locations in which it may be placed is limited. If an icon is "dropped" in a position other than the set managed, it will snap to a different position upon landing. This behaviour cannot be simply explained in terms of the ATTRACTION schema. Within the desktop environment, the grid of points to which icons snap is invisible, the attracting object within the schema is therefore missing and must be inferred by the user for the system behaviour to be understood in terms of the ATTRACTION schema. Even if such an inference is made, the behaviour of an environment intended to be understood in terms of the behaviour of physical world objects will always be unpredictable.

There are some ontological issues raised by snap-dragging, however, that the image schema theory exposes. While the caret is attracted to the cursor and to other attracting objects, its course is deflected by being attracted the stationary artefacts. Attraction is only one-way, however, the caret cannot deflect the path of the cursor, nor can the attractive forces of the CAD objects deflect the path of the cursor. This behaviour would be difficult to find a physical world analogy for — say in terms of magnets of various strengths and objects made of lead or soft iron — recourse to the uni-directional ATTRACTION schema allows a more realistic account of the snap-dragging system to be developed by users.

5.4.3 Case Study 3: The Apple Macintosh Trashcan

The file deletion mechanism employed within the implementation of the DESKTOP metaphor on the Apple Macintosh, the trashcan, is notorious for the problems it causes users, as was discussed in Chapter 3. The difficulty it creates for users that is discussed here is its second use as the means of ejecting floppy diskettes from the disk drive by the user dragging the icon of the disk into the trashcan. This action sequence has been found to cause users distress when they first perform it, many

anecdotes tell of users' feelings that the contents of the disk will be deleted when the icon is placed inside that trashcan. The study of first-time users of the Apple Macintosh reported in Chapter 3 also found that the action sequence does not occur to users, it is not suggested by the metaphor as a means of achieving the task.

Rohrer (1995) describes his attempts to teach the use of the Macintosh to novice users, and reports difficulties arising from a number of different teaching strategies. Rohrer argues that the DESKTOP metaphor is part of a larger PHYSICAL WORLD metaphor from which the notion of removing an object from view can be inherited to explain the use of the trashcan. By indicating differences between the actual system behaviour and the behaviour suggested by the metaphor, Rohrer's students mistrusted the DESKTOP metaphor entirely, and Rohrer claims that they would not generalise from specific cases of system behaviour to the general. This in contrast to claims by Carroll, Mack, and Kellogg (1988) that mismatches can be productive in forcing a greater understanding of the system. Rohrer's second teaching strategy was to provide technical explanations of why the particular action sequence to be performed to achieve the task might have been programmed in the way it was. Instead of users adopting a "design stance" towards the system (Dennett, 1978), however, users were seen to adopt an "intentional stance", and to try to guess the motives of the Macintosh's designer, and adopting a conversational interaction style with a perceived agent within the machine.

Rohrer, in an effort to understand the failure of the trashcan, adopts Smith's (1987) distinction between literal and magical features in user interface metaphors. He suggests that "The magic of a trash can has to do with its being a portal to the beyond in the PHYSICAL WORLD metaphor — the beyond of the landfill, the beyond of the electronic bit bucket, and the beyond of the world outside of the computer." This statement hints at an explanation as to why the trashcan fails, there is a confusion as to which domains the mapping is made between, and the image

schemas underlying understanding of the system. Dragging a disk icon into the trashcan appeals directly to the IN schema, an object is placed within a container and according to the schema should remain within the container. Within the domain of the computing functionality and hardware, however, which following Laurel (1993) and Treglown (1994) should be the target domain considered, the floppy disk is ejected from the disk drive, which can be understood directly by the OUT_1 schema. The Macintosh trashcan requires the user to construct a mapping between two opposite actions, the schema that explains the disk being ejected has no metaphorical mapping in the desktop model world, and is unlikely to occur to users, as found during empirical studies. By requiring an OUT schema to be realised by performing actions that make up an IN schema, the meaning of the operation is the opposite of the way in which it is articulated, it is possible to claim that the task and the trashcan are being ironic. Irony being:

> "...traditionally seen as referring to situations that postulate a double audience, one of which is 'in the know' and aware of the actor's intension, whereas the other is naive enough to take the situation or utterance at its face value." (Gibbs, 1993: 262)

The trashcan is an example of an aspect of a user interface metaphor that can also be said to break the "Invariance Principle" (Lakoff, 1993: 215) which states that:

> "Metaphorical mappings preserve the cognitive topology (that is, the image-schema structure) of the source domain, in a way consistent with the inherent structure of the target domain."

5.5 Conclusions

In the previous chapter, the use of analogies and metaphors in user interface design was examined. Work in this field has demonstrated that analogies often cannot account for all of the functionality and behaviour of a target computer-based system. Where the analogy cannot account for the system image users must either employ inappropriate, possibly superstitious, knowledge or must develop a more realistic model of the system. A supposed alternative to employing analogies is to provide a more realistic model of the system as part of the system image, rather than rely on users to form a more useful and realistic mental model after breakdowns in the analogy and analogical mapping. In this chapter a number of approaches to describing and thinking of mental models were reviewed. Some of these, it was suggested, are inappropriate for describing the knowledge needed to model aspects of the systems described in Chapter 2. Other approaches are found to be those which still also rely on metaphors for understanding of the device. The concept of mental models cannot be ignored, however, the important account of cognition provided by Holland et al. (1986), for example, defines analogies and metaphors as mappings between mental models, or as higher-order mental models in their Q-morphism descriptions of knowledge.

Studies have demonstrated that providing users with a model of the system in terms of the internal components, their topology, and causal relationships between devices has usability advantages. If an attempt is made to model the knowledge that such information is intended to encourage the development of, it is found that breakdowns in the analogy employed in an existing system can be accounted for. These breakdowns, however, can only be accounted for if aspects of the state of the underlying computer system are referred to. This approach also suggests that because computation has a temporal duration, different models are needed to account

for the behaviour shown by the system in the model world in response to user actions, and for the computation that the user actions and system behaviour are intended to be analogues of. That different models are needed to account for the same phenomenon suggests that direct engagement cannot be assumed with on-screen objects when certain performing tasks in a metaphor-based system.

In the following chapter, we present a new user interface design. This system is intended to support tasks that are supported by the metaphor-based systems described in Chapter 2, and it attempts to provide the user with a realistic and useful model of itself. This system is also used to explore the limits and requirements of systems where direct manipulation and engagement are to be supported.

Chapter 6

The Medusa User Experience

"The solid cannot be swept away as trivial and nor can trash be established as solid. It just does not happen."

— Cornelius Cardew, words from paragraph 7 of "The Great Learning" (1968-1971).

6.1 Introduction

The previous chapters have shown that while employing metaphor in user interface design is a powerful technique in attempting to produce usable interactive systems, interface metaphors can also be a source of users' difficulties. The previous chapter showed, though, that metaphor cannot be ignored as a source of understanding. This is the case whether one adopts the idea that understanding of interactive systems comes from mental models, or the idea in the Lakoff/Johnson theory of metaphor that understanding comes either directly in terms of patterns of interaction with the physical world or by metaphorical extension from these patterns. The remainder of this thesis will address the design of a number of new user interfaces collectively termed Medusa. The first Medusa system is described in this and the following

chapters. The first Medusa system adopts the conclusions of traditional, comparison, theories of metaphor and the qualitative process theory (QPT) analyses of tasks in direct manipulation tasks. The second Medusa system assumes the Lakoff/Johnson theory of metaphor understanding in its design. This chapter sketches the intended user experience of the first Medusa system, Chapter 7 discusses the design rationale of the first Medusa system, and Chapter 8 presents the results of usability evaluation of the first Medusa system design. A revised Medusa system design, that assumes the Lakoff/Johnson theory of metaphor understanding, is discussed in Chapter 9.

6.2 Basic Criteria that the Medusa System Should Satisfy

The Medusa system is a user interface design that takes into account the criteria listed below:

- Simple basic tasks, involving the functionality of a computer's operating system and file management system, which will be performed at some time by every user of the system, should be supported.

- A conceptual model of the system should be given to the user which can consistently support data file types which are not naturally supported by existing metaphors (for example, sound and video fragments).

- A conceptual model of the system should be provided to the user which presents a low overhead when learning the system, yet provides the advantages of possessing mental models when performing novel tasks and understanding unfamiliar system behaviour,.

- It should be ensured that mental models of the system formed by users are consistent and that attention is paid to the consistency of the behaviour of on-screen objects and the actions that may be performed on them.

- It should be ensured that state feedback is timely and appropriate following an awareness of design solutions suggested by work on formal models of interactive systems.

- It should be noted that metaphor and analogy play a major part in learning, understanding and interaction with the world and cannot be ignored. A design should take into account the role of metaphor and analogy in learning and using user interface software. A design should, however, be aware of the problems that metaphors and analogies in the model world cause the user as well as those that they solve.

The details of the design rationale underlying the first Medusa system are provided in the following chapter. In this chapter a sketch of the intended user experience when using Medusa is provided.

6.3 General Layout of the Medusa Display

When starting the first Medusa system, its user interface is unlikely to present any initial surprises to a user familiar with common implementations of the DESKTOP metaphor, or WIMP interface style. The user will see a 2D window that occupies the entire area of the display(s) connected to the central processing unit's graphics hardware. This window, the *root window* as it is termed in the X window system, or the *desktop* in the DESKTOP metaphor, is always the rear-most window, the user cannot place any windows behind it in the stack of windows that occlude others. As can be seen in Figure 6.1, though, the root window itself is much the same as other

WIMP systems, it serves as an area on which file icons may be placed by the user as reminders, or to be used in their immediate tasks.

The major difference between Medusa and other WIMP interfaces that users will notice is the on-screen graphic which can be seen in the top-right hand corner of Figure 6.1. This graphic provides information about the status of the underlying computing system, of data structures relevant to the performance of the current and subsequent user tasks, and of additional devices to which the processing unit is connected. This graphic is also the source of "meta-objects", meta-level representations of on-screen objects that the user employs directly when performing tasks but which themselves lack any means of having their attributes and behaviour modified by the user. These meta-objects acknowledge what Dourish (2001) calls the *inflexible obtrusiveness* of most graphical user interfaces that makes invisibility an unobtainable goal of many interface designs. Instead Medusa adopts a design approach discussed by Thimbleby (1990: 229) which Karl Popper termed *Berkeley's Razor*[1]. Berkeley's razor is the notion that "All entities are ruled out except those that are perceived." Any information, or mechanism, that is required to perform tasks in Medusa, or that becomes apparent at a point of breakdown of its usual behaviour is made apparent to the user. The meta-objects shown in Figure 6.1 and their behaviour are an attempt in the first version of Medusa to realise this notion.

6.4 Performing Basic Tasks in Medusa

The major source of input to Medusa generated by the user is via a pointing device capable of generating selection information. With existing common computing technologies, and with the technology assumed when considering prototypes of the Medusa system, this pointing device is likely to be a mouse. However what is

[1] Popper means this to be a "sharper" version of Occam's razor ("plurality is never to be posited without need"). Berkeley's razor is named after the philosopher Bishop George Berkeley (1685-1753).

important to Medusa is the *design space* of the device (the range of data that it is capable of generating and how this maps into the underlying software's data structures) not the family of devices from which a particular device is chosen. Most tasks, as will be explained in further detail in the following chapter, are selection tasks. The data required from the user via an input device therefore need not include paths of points on the display, such as would be generated by polling, sampling, or logging events generated as a mouse, say, moves. This allows us to consider in the final chapter possible implementations of Medusa on small screen devices and personal digital assistants (PDAs).

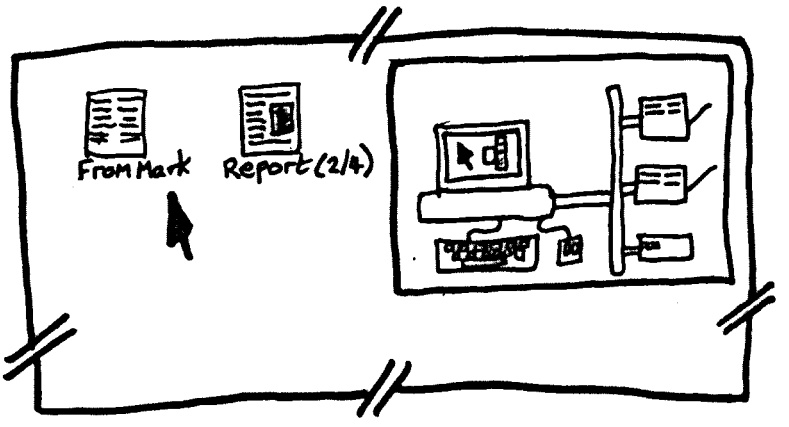


Figure 6.1 General layout of the Medusa display

6.4.1 Using the Toolbar

The first version of the Medusa system uses an *object-message* syntax for interaction. All but a very few on-screen objects respond to messages that affect the attributes of objects. On-screen objects, however, will be members of very different categories (or classes) and will respond to different sets of operations that bring about changes in their states. A well-known problem that HCI addressed early in the design of graphical user interfaces was the problem of interaction modes, where systems respond in different ways to the same user input depending on the current state of the device. Alan Kay, as was mentioned in Chapter 2, is said to have devised

overlapping on-screen windows in part to resolve the problem of modes (Bardini, 2000). The difficulty of having to account for modes using (possibly overlapping or mixed) metaphors was discussed in Chapter 4. Thimbleby (1990) states that it is inaccurate to speak of *modeless* systems, for user input to be interpreted at all by the system it must possess at least one mode. Instead designers, Thimbleby argues, can strive for *low-mode* systems. This striving reflects what Bardini (2000) characterises as the Xerox PARC tradition in user interface design. By contrast, Engelbart's NLS system, according to Bardini (2000:118), "...multiplied ... discrete states or modes into so many exclusive conditions of the user's activity. To tap into the functionality of a given command, the user needed to establish a certain configuration of preliminary commands to put the system into a specific mode in which the needed command was available. In such a system, the user had to memorize where he or she was in the hierarchy of commands and modes. The interface was a kind of maze, often requiring backtracking to access new functions and commands." Douglas Engelbart intended that his NLS would be used by experts and knowledge workers. The chord keyboard required to navigate between modes proved, however, less usable by more casual and infrequent users (termed "human beings" in David Canfield Smith's somewhat mocking description[2] of the comparative usability of the chord keypad and the use of the mouse in the Xerox Star).

If different categories of on-screen object respond differently to similar user input, as they do in Medusa, then Medusa is modal. What Medusa does, however, is to make interaction with all on-screen objects simple and consistent so that the modes are not apparent, or are regarded as no more complex than menu-based interaction. A toolbar presents the options available in the current state that can be applied to an on-screen object indicated by the user. The appearance of a toolbar in response to the

[2] Commentary to a video recording of the final demonstration of the Xerox Star held at Xerox PARC, July 1998.

user invoking it is shown in Figure 6.2. This illustration uses the storyboarding conventions of Katz (1991). When the toolbar is visible, the option that the user subsequently clicks on will be the option or command that is applied to the object.

When the toolbar appears, it partially occludes the icon for which it was invoked. This is meant to reinforce association of the toolbar with the icon, and to indicate (as with the Magic Lens user interaction technique described in the next chapter) that the message passes through the toolbar to the icon behind it.

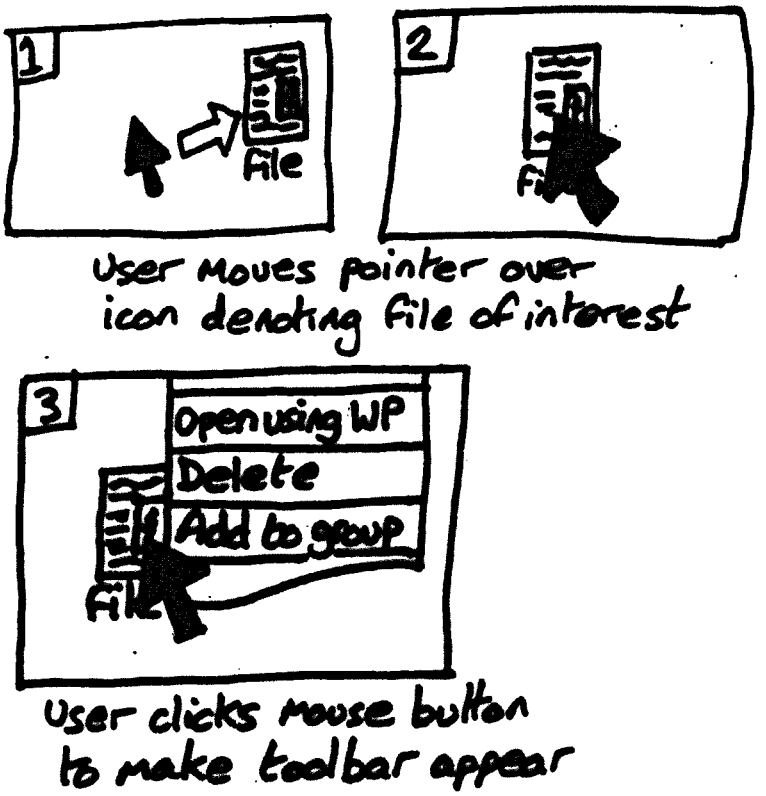


Figure 6.2 Invoking the toolbar for an on-screen object

6.4.2 Collections of Objects

Computing systems of the sort surveyed in Chapter 2, and of which Medusa is intended to be one, usually provide facilities for organising data files produced by

application software into spaces from where they can be retrieved by users. Users often need to create collections of data files that share properties or which are required by users to perform their immediate tasks. In Chapter 9 some of the arguments over the need to archive and organise information are surveyed, and alternative user interface designs for file organisation are compared using the Lakoff/Johnson theory. In the first version of the Medusa system, however, a comparatively simple interface design is adopted which follows from the QPT analyses presented in the previous chapter.

In the first version of Medusa, collections are of two possible types, long-lived, or short-lived. A long-lived collection, which would be formed from files and directories in command-based system file spaces, or from files and folders in the DESKTOP metaphor, is denoted in Medusa by a container. A container (shown in Figure 6.3) has a simple icon that shows relevant properties of the underlying implementation in the operating system's file system. Containers can be empty, or they can currently contain files and other containers. Users' notions of containment are particularly important for understanding user interfaces. The simple icon design makes the reliance on ideas of containment (which can be seen in the QPT models in the previous chapter) apparent. The file and folder method of organising data in the DESKTOP metaphor also relies upon ideas of containment, as will be discussed further below, but the metaphor is a weaker one than the "container is a container" idea employed in the first version of Medusa.

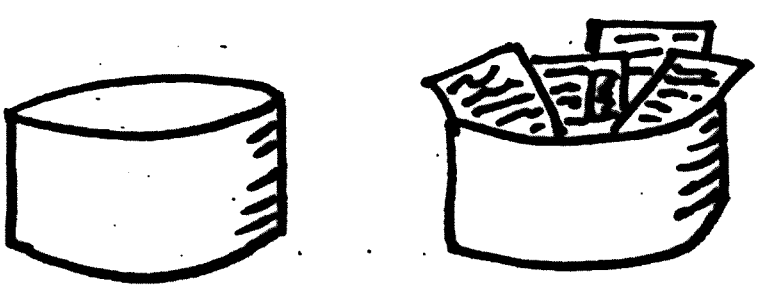


Figure 6.3 Collections of objects — containers

Most collections of objects are long-lived, and are stored within the hierarchy that file systems usually allow the user to construct. Occasionally users, though, wish to construct short-lived collections. These collections usually comprise files that reside within the same window, or that are all on the desktop. In most DESKTOP metaphor systems, two methods are supported by the interface to allow these collections to be constructed. One method is to allow the user to lasso a number of icons by pressing the mouse button while the pointer is in an empty region of window in which the icons lie and by dragging the pointer to another empty point in the window. While the pointer is moving, a rectangular bounding box (the lasso) is drawn and redrawn so that one corner of the box lies on the point at which the mouse button was pressed, and the corner opposite lies on the current location of the hotspot of the pointer. The second method is to select the first file by clicking on it, and then selecting subsequent choices by modifying the mouse button click, by using a different mouse button or by holding down the shift key on the keyboard while clicking on the additional files.

These methods both follow from the idea the notion of the currently selected object. Medusa, for reasons explained in the following chapter, does not adopt this notion in its interaction style. Medusa instead allows short-lived collections to be constructed via the toolbar options **Add to Group** and **Remove from Group**. This is shown in Figure 6.4. If the toolbar is invoked for a file when that file is currently member of a group then the toolbar's contents will be different. Rather than contain the operations that can be applied to the object itself, as would be the case for an individual object, the toolbar will instead contain the operations that are meaningful when the file is considered as a member of a group. The difficulty lies, as the following chapter explores, in ambiguity of reference and of deitic reference, determining which object the user meant when indicating an icon as the recipient of a message. Because the same region of on-screen space (the icon) can be interpreted

as either an individual or as a metonym for the group of which it is a part, the toolbar must display options that take into account both of these cases. The toolbar must allow the user to resolve the ambiguity. An example of the sort of toolbar that might appear when an icon is part of a group is shown in Figure 6.4. Chapter 8 considers more fully the alternative low-level sequences of user-generated events that might be adopted to support the basic Medusa interaction tasks. While the low-level tasks for interacting with groups described above are consistent with the Medusa interaction style, the lasso method can also be supported, as a synonym for repeatedly adding files to a group. This can only occur in implementations where the input device used can differentiate between PRESS and RELEASE events in deciding if an event is the beginning of a lasso task, or should be interpreted as invocation of the toolbar.

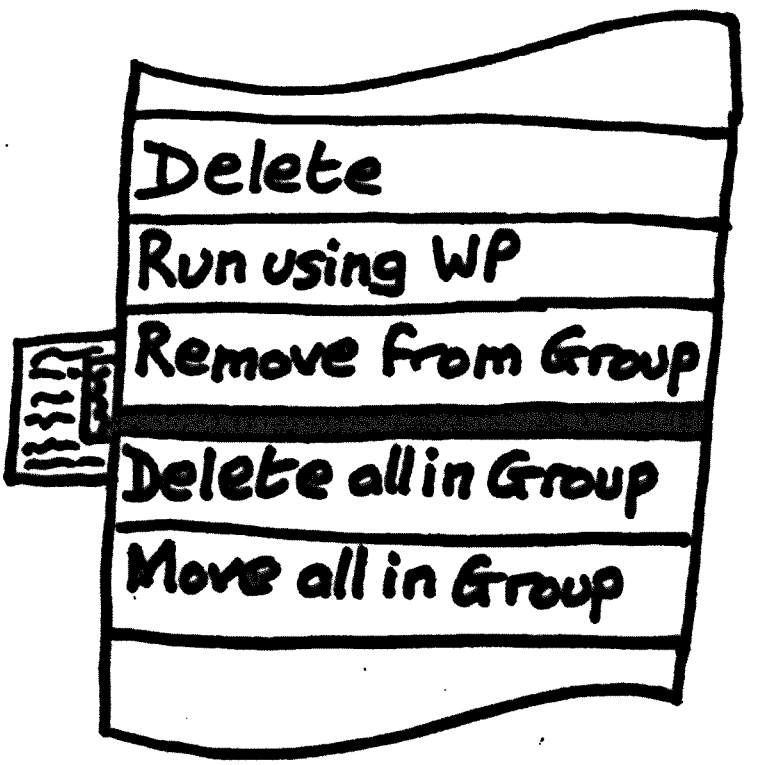


Figure 6.4 A Toolbar for a Group

6.4.3 Moving Files between Containers

The QPT models developed in the previous chapter revealed a considerable mismatch between the behaviour of objects in the model world and the behaviour of the corresponding objects in the underlying software. This mismatch cannot be explained easily by mapping metaphorically between the domains, and it also reveals a breakdown in *direct manipulation* in the model world (Chapter 9 discusses Lakoff's definition of this term which is adopted instead of Shneiderman's definition in later thinking about the design of Medusa). Far from acting on the actual objects of interest, the implementation of the drag-and-drop interaction method for this task ends up with the interface, not in a state in which the underlying system is actually in, but a state in which the system is expected to catch up to — what you see is what you may eventually get. This interaction task can also give rise to semantic errors, the user can drag a file to a container into which it cannot actually be placed, perhaps because the disk volume it denotes is full or locked and read only. The Medusa action sequence for moving files between containers borrows more from the pragmatic implementation of the Xerox Star than the DESKTOP metaphor, and is storyboarded in Figure 6.5.

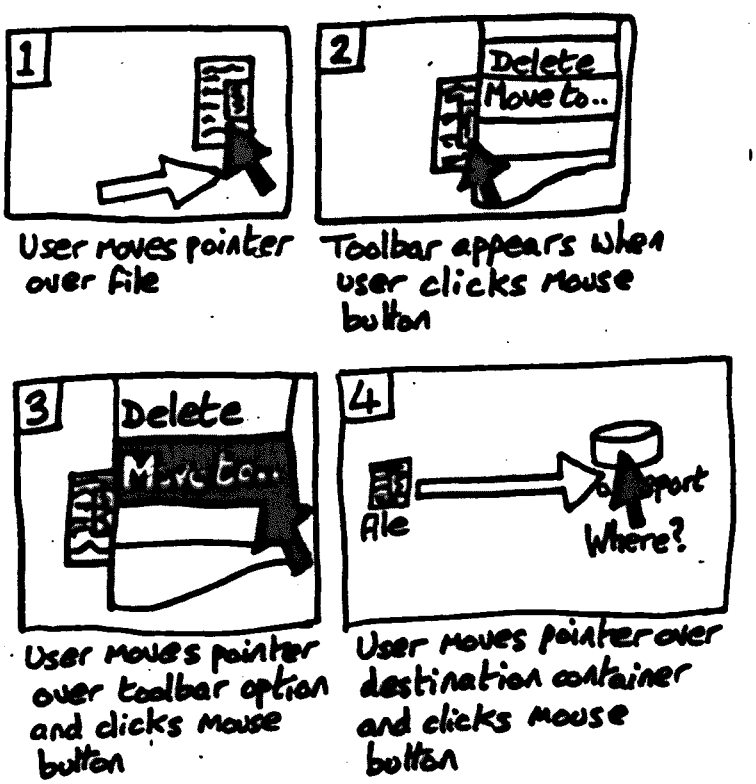


Figure 6.5 Placing a data file into a container

6.4.4 Deleting Files

The study of first-time users of the Macintosh described in Chapter 3 showed, confirming other reports, that the TRASHCAN metaphor is problematic. This was found to be the case even for the trashcan's first use as a mechanism for deleting files. The comments made by the subjects in the study reported in Chapter 3 are echoed in users' comments quoted other reports, for example: "See the recycle bin? Does someone come round and empty it?"[3] The trashcan is not a file deletion mechanism in itself, it is instead a directory in which files can be stored while the user decides whether to delete its contents, or retrieve them. The file deletion task is made simpler by having an **Empty Trash**, or **Empty Recycle Bin**, command

available on the task bar at the top or bottom of the screen. The location from which the files are deleted is intended to be obvious, the trashcan. A failing of the TRASHCAN metaphor is that even when a real trashcan is emptied, its contents exist until incinerated or irretrievably lost to a land fill site or refuse tip. The TRASHCAN metaphor as implemented in existing computing systems offers no way of attempting to retrieve a deleted file. Rather than attempt to provide a retrieval mechanism within Medusa, instead, as with existing systems, a specialist application will be required to recover deleted files. As was shown in Chapter 5, though, some implementations of the trashcan might have far more complex behaviour than being a store of files, the store being simply the recipient of the **Empty Trash** message. As was seen in Chapter 3, the TRASHCAN metaphor does not seem to suggest, though, how files can be retrieved from the trashcan. Other metaphors for file deletion, such as the "black hole" in ARK are more complex than the trashcan while being weaker metaphors for the actual deletion mechanism. The solution to allow files to be deleted from the Medusa file space is simply to have a Delete option appear on the toolbar associated with a file. Undeleting a deleted file is a task that can only be performed for a short period of time until the data blocks on the disk that it occupies are recycled by the operating system to store new files. There is nothing within the Medusa design to prevent a suitable recovery application to be used to retrieve data from the disk, but the design of such an application will not be considered further. While the data blocks making up the file have not been corrupted, it may be possible to undo the deletion operation. The issue of undo within Medusa is discussed further in Chapters 7, 9 and 10.

---

[3] Telephone call to a technical support help line, reported in The Editor, supplement to The Guardian newspaper (13th April 2002).

6.4.5 Interacting with the Root Window

Many user interfaces employ a menu bar to allow commands to be performed. A menu bar is usually laid out across the top of the screen and contains a hierarchy of commands on pull- or drop-down menus. A menu bar is a permanent fixture on-screen while the windowing system is running. Menu bars, though, enforce the idea of the currently selected object, and require the user to move the mouse perhaps a considerable way to reach the command needed on its menu in the menu bar.

Task bars are simpler menus that are located at the bottom of the screen (in the case of the Macintosh command strip, the Microsoft Windows 95 task bar, and the strip of large icons denoting commonly used applications placed along the bottom of the Macintosh OS X display). Task bars usually contain functions or links to applications that can be applied in any context without first selecting a file to apply them to. The user may still need to move the mouse some distance to reach the task bar. In Medusa the root window is not a desktop, it is just another active object and so a toolbar can be invoked which can send messages to the root window itself. The root window's toolbar can, for example, contain commands to end the user's interaction session, or shut down the workstation (exploiting the sorts of SPACE for TIME metaphors discussed in Chapter 9) and can also be used to invoke commonly used applications. A similar device can be seen on some WIMP systems which do not implement a strong version of the DESKTOP metaphor, but this is an additional interaction style that users must learn, and often do not predict or imagine, to the

traditional use of pull-down menus. In Medusa, having a toolbar apply to the root window is entirely consistent with the key interaction style.

6.5 Breakdowns

Where user interfaces, particularly those that are based upon interface metaphors, cause users considerable difficulties is at points of breakdown — where their behaviour suddenly differs from their normal, usual, or expected behaviour.

6.5.1 Hardware Breakdowns

Most hardware failures make a computing system inoperable. There are other failures, though, particularly of networked devices, that make tasks impossible to complete, or which give rise to unexpected system behaviour that the user must attempt to interpret and remedy. This can be difficult because networked devices may not be directly visible to the user. Even if a device is in the same room as the user, it may not be able to understand the source of the breakdown from a change in its outward appearance. The on-screen graphic shown in Figure 6.1 allows a number of breakdowns to be easily observed via colour coding and other feedback of the sort proposed by Polson which were described in the previous chapter. Network failures between the central processing unit and devices such as printers or file servers can be indicated easily. Depicting the devices as icons on the root window makes it possible for the user to examine their state. For example, the user can easily determine if a printer is out of paper, or to judge how many other printing tasks must be completed before the user's document will be ready to be collected. There is nothing to prevent

background or ambient audio, such as the sort employed in ARKola, to also communicate this information, and the on-screen graphic shown in Figure 6.1 provides a *handle* to indicate to the user the source of the sonic information.

6.5.2 Buffers

Breakdown from the expected, usual, system behaviour to unusual system behaviour can arise from the use of buffers. Buffers are data structures that store data for a short time until it can be processed by the application for which it is destined. Usually buffers are unnoticed by the user. The data that the user generates, such as characters typed at the keyboard, are processed seemingly instantaneously by the application in focus. If the system load increases, though, there may be a perceivable lag between the typing of characters and it appearing inside a text editor window, say. The user thus becomes aware of the existence of the buffer. The first version of the Medusa system deliberately sets out to make the user aware of the existence of buffers. Following a design proposed by Dix (1991) when characters remain in the buffer for a perceivably lengthy delay without being consumed by the application, a visualisation of the buffer, shown in Figure 6.6 appears. As the user types further characters, these appear appended to the end of the buffer's contents. Figure 6.6 makes this appending of characters apparent even when the user presses the backspace key to delete a key pressed in error. The application will interpret the delete key in the way that the user expects, the buffer cannot. When the application processes characters, they are removed from the buffer. The common use of the work "consume" to describe an application processing a character removed from the buffer leads Dix to refer to this design as the *munchman* (aka Pac Man) buffer. The

Pac Man metaphor, however, possesses considerable conceptual baggage and is not fully adopted by the Medusa system.

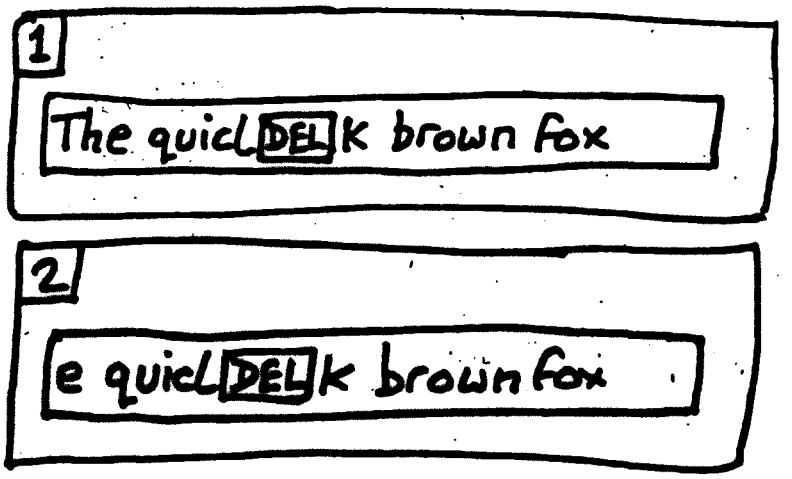


Figure 6.6 Visualising the Medusa keyboard buffer

6.5.3 Predicting Breakdown

The munchman buffer and the use of colour coding on the computer-computer schematic indicate points of breakdown, moments when the look and feel of the model world deviate from their usual behaviour. At such a moment the user is thrown into a state where they must consciously diagnose the system in order to predict the outcomes of further actions. In a literal user interface metaphor, the breakdown cannot be explained in terms of the metaphor, and the possibility that a breakdown is forthcoming cannot be made apparent to the user. In the previous chapter, the idea of the load on the processor as a source of breakdowns in the normal behaviour of on-screen objects was discussed. Early graphical user interfaces to Unix such as Sun Microsystems' OpenWindows provided a small utility program that could represent the current processor load as a dial display, or could plot the

recent history of this measure as a graph. A number of windows systems developed since have included this information on-screen or have had similar utilities developed for them. Medusa too can display this information, perhaps overlaid on the processor box image in the computer-computer schematic, or in a separate display elsewhere on-screen. A simple measure of the processor load, however, does not allow the user to predict all breakdowns, or to excuse them when they occur.

The user's sense of spatiomimesis is that the on-screen pointer is an extension of themselves in the model world that exactly tracks the user's movement of the mouse. Breakdowns in spatiomimesis can be highly disruptive, the feedback process that underlies Fitts' law and allows users to move the on-screen pointer accurately to hit the (sometimes small) on-screen buttons is disrupted and users may inadvertently click on a target they did not mean to. Mouse-ahead facilities can also cease to behave as expected, it being unclear where on-screen the windows system records the pointer as being when a mouse button click event is generated. Most windows systems operate on a repeated "read next event from event queue → process event" cycle. If the processing of a mouse movement event, say, takes too long, then the next mouse movement event in the queue will not be processed until after the deadline of 50 milliseconds by which the display should have been updated in order to maintain the illusion of animation. The design of a revised version of Medusa called Medusa-τ, discussed in Chapter 10, is intended to overcome this problem by application of real-time programming methods following detailed specification of the temporal behaviour of the user interface. This approach is intended to remove, where possible, the problem. In Medusa, the intention, where possible is to explain the problem in order to account for breakdowns and to allow the user to plan

subsequent action in light of the system's deviation from its normal behaviour. Rather than use the approach of changing the pointer icon to a symbol that is a poor metaphor for its underlying *simulated state of understanding* (Pérez-Quiñones and Sibert, 1996) — the rationale for rejecting this common approach is presented in Section 7.2.9 — colour coding is used to indicate, via the pointer itself, the time taken to process the last event on the queue. For a single lengthy event, the problem of distracting the user should not be a considerable problem. Where the pointer icon changes to indicate a breakdown in normal event processing, often designers choose to *not* change the pointer's shape when it would switch back and forth between different shapes too rapidly. A change in pointer shape at all other times indicates a change in mode, changing from a traditional pointer (↖) to a double-headed arrow (⇔), for example, indicates a point on the vertical edge of a window which can be used as a drag point to adjust the width of the window. Changing the pointer's shape in response to delays in event processing suggests that the mouse has changed mode. In fact the mechanism of the mouse event processing system is the same, but with different temporal behavior. Medusa should make relevant aspects of the mechanism and the conditions under which it is operating visible, not suggest that a different mechanism is at work.

6.6 Conclusions

This chapter has sketched the intended user experience that Medusa should offer. The details of the design rationale underlying the user experience are given in the following chapter. No working prototype of Medusa exists but this does not prevent usability analysis from being undertaken. Usability evaluation of the first Medusa system using a low-cost *usability inspection* method is reported in Chapter 8.

Chapter 7

The Medusa System Design Rationale

"It's like the mozzarella cheese on a good slice of pizza. No matter how far you pull the slice away from your mouth it just gets thinner and longer but never snaps. Of course you could always just eat your pizza with a knife and fork, but I think this is clearly what's known as 'pushing the cheese analogy'".

— Jerry Seinfeld (1995) *SeinLanguage*, Bantam Books.

7.1 Introduction

In this chapter, the motivations behind the Medusa user interface design are discussed. It is judged that systems are needed that provide access for novices to the functionality provided by operating systems to support a range of tasks that overcome the difficulties with existing metaphorical model worlds , while allowing the user to develop a useful mental model of the system. It is hoped that the Medusa system will not be subject to breakdowns, erratic behaviour of on-screen objects, and users' misunderstandings, to the extent that existing metaphor-based user interface designs are. Details of the Medusa system design and the intended user experience were presented in the previous chapter.

7.2 The Medusa System - Version One

Having stated the criteria that the Medusa system is intended to satisfy, having introduced the means of modelling the system, having detailed some of the models that the system image should evoke, and having sketched the intended user experience, we now describe the Medusa system design rationale in some detail.

7.2.1 The Workbench

In the desktop metaphor, there exists a root window (as it is termed in the X window system), a window that occupies the full area of the display(s) connected to the workstation, which cannot be resized or moved, and which always lies behind all other windows. This window is what is termed the *DESKTOP* in the desktop metaphor. The desktop is meant to be the analogue of its real-world counter point, an area upon which tools and documents may be placed while the office-worker carries out their tasks. The electronic desktop, as has been previously mentioned, differs in some important respects from its real-world counterpart. The trashcan, for example, sits *on top of* the desk rather than beside it, as do file storage containers such as filing cabinets.

Donald A. Norman, for one, prefers to think of the root window as a *workbench* rather than an electronic desktop. The workbench is an area provided for planning and the storage of icons while sub-goals are suspended in favour of more immediate tasks that alter the state of the file system. In the DESKTOP metaphor, the root window creates difficulties that cannot be accounted for in terms of the metaphor. It

has been asked[1], for example, what it means for an application program to be moved onto the desktop. In the first version of Medusa, as shall be discussed below, on-screen objects are links to files, the workbench is treated in the same way as a directory or folder, links may be placed on the root window and moved within the window to suit the needs of the user. Medusa treats the root window as another rendering of a directory listing file, but one of fixed size that cannot be scrolled. It is hoped that this interaction style is clear from the discussion in the previous and following chapters.

7.2.2 Objects in the Model World

The first version of the Medusa system adopts the critique of existing metaphor-based systems and models of systems arising from QPT modelling of the physical world as applied to user interfaces, as discussed in Chapter 5. Thus it is envisioned that the behaviour of the model world should be explicable in terms of simple processes acting on the on-screen objects and that these processes should also be applicable to the underlying software objects by a simple analogy. Medusa should therefore be unlike the situation in existing systems where often no mapping can be found between processes acting on on-screen objects and those acting on underlying software objects. This model of system design, the *product-oriented* view in the terminology of Andersen (1997), requires that analogies that follow from the underlying software be sought in the real world to describe the model world. The structure of a metaphor is therefore that of Laurel's (1993)description above, where the metaphor mediates between the model world and the underlying software. In such an approach — where the model world employs an existing work language, or

---

[1] This question was brought to my attention in a discussion with Professor Alan J. Dix in the winter of 1991.

resembles some aspect of the real world — , we acknowledge that the work language risks being imposed upon the user (Brock, 1996).

Objects in the Medusa model world are members of classical categories. That is, membership of categories is determined by the necessary and sufficient attributes possessed by objects to be a member of a particular category. Category membership determines the operations that may be performed upon on-screen objects. As the Medusa user interface is object-based, the categories of Medusa on-screen objects can be said to form a class hierarchy as some objects have more attributes than others yet are similar to members of other categories. This class hierarchy is depicted in Figure 7.1.

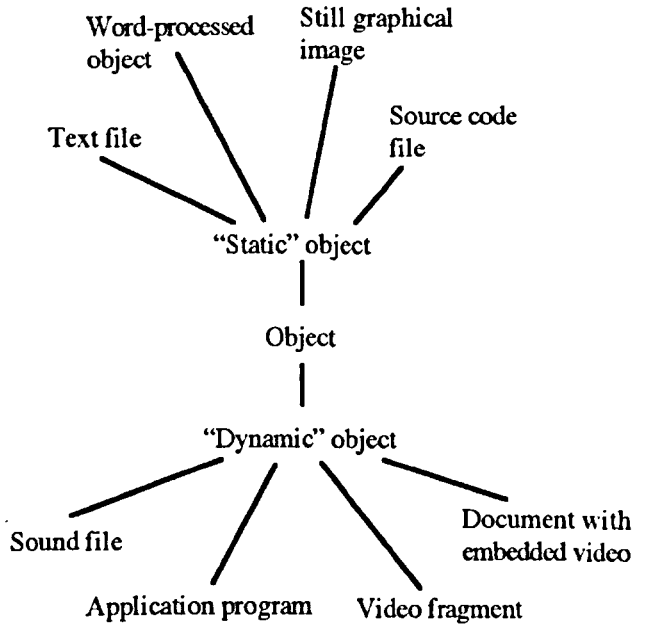


Figure 7.1 The categories of Medusa system version one on-screen objects

The principle distinction between categories defined in Medusa is between static objects and dynamic objects. Static objects are those that can change as the result of software tools being applied to them but which produce a file whose presentation does not change, such objects would include documents, text files, and still pictures. Dynamic objects are those whose presentation changes with time, such objects

would include video fragments, sound and music files, and multimedia documents with embedded sound and graphics. This distinction between dynamic and static objects is made so that the processes that alter or act on the different categories of objects are made apparent. It makes no sense to try to print a dynamic object such as a video fragment for example, but by using a suitable software tool a static object, a still image consisting of one video frame, for example, can be produced which can be printed. This distinction was also made subsequently by Fitzmaurice, Ishii, and Buxton (1995).

In many systems, icons are assigned to objects in a computer's file space according to the suffix placed at the end of the files' names. All files with a ".c" suffix, for example, can be depicted with the icon denoting a C programming language source code file or be assigned an icon denoting the application used to create the file. This approach has a number of drawbacks. Firstly, it may be misleading. For example some files saved in the GIF graphics interchange format may not be still images, but may be simple animations comprising a sequence of frames that are displayed in a loop. Some word processors, in addition, permit the creation of documents with video and sound fragments as elements of the page. Hence one might assume that a file might be printed by extension of one's previous experience of using files of that type, whereas it cannot actually be printed. The common depiction of files of a certain type presents additional problems including the perception of a limit to the uses of a file or to the number of software tools that may be applied to read and alter files. Opening a file will cause an application to run and load the file opened, the application run will typically be denoted by the file's icon although many other programs may also be capable of using and modifying the data contained in the file. The same can be said for icons within a typed file system, such as the Macintosh Finder, where an ontology of objects exists with associated (but possibly modifiable) icons, rather than where rules employing a filename's suffix are used to determine what icon design should denote a file.

The approach adopted in the design of icons that denote objects in the Medusa model world is one termed *self-representing* (Treglown and O'Shea, 1993), a notion that mirrors, but which was developed without knowledge of, the notion of self-identifying objects[2] employed by Putnam (1981). A self-identifying object is one that evokes only a single concept or thought token when perceived. Self-representing objects are those that use, so far as practicable, the final form of the file in the generation of a suitable icon to denote the file. While not simply adopting icons denoting files of a particular category, it is hoped that category membership can be determined from the icon's design. The notion of self-representing icons also adopts the product-oriented approach to metaphor (Andersen, 1997), in keeping with the Medusa system design where objects in the model world are designed according to some metaphor to account for features within the underlying computer system .

Text Files

The Medusa system is intended to make apparent to the user relevant aspects of the computing system The device topology and the nature and state of data structures will be visualised where such information is required to provide a full account of the behaviour of the computing system. All files in a computer's file space consist of a sequence of bytes represented in the physical medium of the disk drives connected to the processor. The ways in which the bytes of information are interpreted by the application tools used by the user are seemingly contrary to the idea of visibility. What is stressed in the on-screen depiction of files is the final form of the data and the ways in which the data may be manipulated rather than the structure of the data that make up the files.

---

[2] Putnam cites David Wiggins (1980) *Sameness and Substance*. Blackwell, Oxford, as the origin of the idea of the self-identifying object.

A simple text file is deemed to be a text file as the result of the file being interpreted by a suitable application program. The application program interprets each byte or word in the file, using an encoding standard such as ASCII, as an alphanumeric or a special control character. A depiction of a text file should ideally depict the final form of the data in a way that means most to the user. It, should also aid in the tasks of locating the file among the icons visible on-screen and of identifying a particular instance of a type or class of file among a number of files of a similar type. Where files created by an application are depicted by the same icon, or where the icon is assigned by the window system according to the suffix on the file's name, one finds icons such as those shown in Figure 7.2.

Figure 7.2 Typical text file icons

Such icons only depict category membership and additional information such as the file's name, yet perhaps its version history and additional comments (supplied possibly by another user) stored with the file, may be required by the user in order to uniquely identify the file. The design of icons may be improved in order to ease performance of location and identification tasks. Experimental studies (Ark, Dryer, Selker, and Zhai, 1998) show that ecological icons, those that closely resemble objects in the real world, can assist with location tasks. Ecological icons, however, by resembling real world objects, are more appropriate to systems based exactly on the metaphors adopted and on the use of metaphor questioned earlier that we shall eventually reject below. Instead icons less realistic than those termed ecological, but richer than the typical icons shown in Figure 7.2 are adopted. Instead of a still image, which denotes only a single page, or the presence of a number of pages in the text

file, motion icons, or *micons*, may be employed to provide a richer icon. In a micon (Brøndmo and Davenport, 1989; Baecker, Small, and Mander, 1991) a sequence of small pictographic symbols is cycled through frame by frame, when the last frame is reached, the first frame is displayed again. Each frame of the micon is an icon. Brøndmo and Davenport (1989) use micons to denote video fragments, a subset of frames from a piece of digital video where each frame is shrunk in to icon size, these fragments being links in a hypermedia network to other nodes containing relevant full-size video sequences with accompanying soundtrack. Baecker, Small, and Mander (1991) use micons to represent simple actions within application programs that denote how to bring about a simple change in state in another artefact produced using, or maintained by, the application.

A micon depiction of a text file may therefore consist of a sequence of icon frames, where each frame is a page of the document shrunk to icon size. While unique identification of the text file is unlikely to be possible from the micon itself, clues may be obtained from the superficial structure of the document as to the document's identity and may distinguish it from other text files or previous versions of the file. Such a strategy for icon design is not without problems, however, and does pose questions that require investigation and answering from appropriate theory and experimental work. The size of a page, for example, meaning the number of lines of text that appear on the page in a simple text editor application can be a fixed integer., More often it is a function of the physical size of the paper currently selected in the printer and of the fonts and number of lines making up the text file. If the preferred configuration of the printer is changed, then the final appearance of the text file will change. , If changes to text file micons are propagated throughout the file space, then recognition of a file being sought will be confused as its appearance will have changed since the user last altered the file's contents. A solution to this problem, one often adopted, is for the preferred printer configuration to be an attribute of the *document* and not the printer. This contrasts with the photocopy metaphor employed

in the Xerox Star system, where one expects the size of the paper that the copy will appear on to be part of the photocopier's state.

Documents

Documents are seemingly similar to text files, but typically make greater use of more complex formatting facilities. The same principle of creating icons to represent text files can be applied to represent documents. Documents, however, may not solely be "static" objects in the Medusa on-screen object ontology depicted in Figure 7.1. Word processing applications often allow pages to contain, in addition to still graphical images, sound clips, video fragments, inclusive links to data created by other classes of application such as databases and spreadsheets. They can also include links to data in other documents where changes to the linked data will propagate to every document that includes it. Where part of a page is a "dynamic" object, then what it means to print the document must be considered.

The approach to generating icons denoting a document can be borrowed from that of the approach for depicting text files, where each page of the document is used to generate a frame of a micon. Where a page contains a video fragment, a micon will appear within the frames depicting these pages. Such micons within micons will only decrease the possibility of uniquely identifying a file from its icon, such will be the loss of information in further reducing the information contained in the video fragment. Such micons will, though, aid the user in telling a micon from others of the same class of file. Again, other information is required in addition to an iconic depiction to uniquely identify a file, the form of this information will be considered further below.

Programming Language Source Files

Computer program source files are often are depicted by icons similar to those that denote text files. While source files *are* text files, a sequence of alphanumeric characters, a question exists as to the most useful final form of the file's contents when it comes to choosing a suitable icon. When printed, similar results will be produced to that of performing the task of printing a typical text file. To the programmer however, splitting the file into pages, each page forming a frame of a micon, is less meaningful a level of granularity for abstracting the file's contents than others that could be suggested. When programming, moving between pages is a less frequent task than scrolling the text of the source file until the class, method, rule, variable declaration or procedure sought is found. Rather than cycle through pages of the file, a scrolling micon would be a better representation. The loss of information that occurs when a legible full-sized page is reduced to the size of a typical icon remains a problem. Considering Brooks' (1983) "beacons", indicators for the meaning of a computer program, it can be seen that prologue comments, variable structure and label names, interline comments, indentation or pretty-printing, and subroutine structure contribute greatly to interpreting an unknown program source file and deciding upon its functionality. Many of these beacons are likely to survive when forming an icon, even though the size of the program text is reduced until the text itself becomes illegible.

Picture Files

It is already common for graphics application programs to allow a preview file to be created, which is an icon of the image created using the application. Seemingly a picture file, or still graphics image, this presents few problems. In the Medusa ontology it is a "static" object which can be easily printed and which is subject to tasks that alter its location within a file space in the same way as all on-screen

objects. The approach of assuming that all graphics files, irrespective of the encoding method used to interpret the bytes in the file space as an image, can be simply printed if a suitable printer is available to the system is however, false. A particular form of GIF file, employed by sites on the world-wide web, allows a GIF file to consist of a number of frames which tend to be interpreted by world-wide web browsers as micons, frames are cycled through in sequence while the image is visible through the browser's window. Such files would therefore be termed "dynamic" and single frames would have to be isolated from them before printing would be a task allowed by the system.

Video Fragments

Following Brøndmo and Davenport (1989), it is suggested that video fragments be depicted by micons within the Medusa system's model world. In Brøndmo and Davenport's Elastic Charles hypermedia system, the history and geography of the Charles river that separates Boston, Massachusetts from Cambridge, where the the MIT campus is located, can be explored. Micons are used to depict links to relevant video fragments within the overall hypermedia structure that provide additional information relevant to the concepts presented on the current page of information, image or video. Within Medusa micons are used in the same way to denote a digital video fragment, and to acts as handles to the data files that contain the encoded video data. A frame of the micon is a shrunken version of a frame of the video. The micon as a whole is made up of is made up of a number of frames of the video fragment reduced to icon size. The frames of the micon are then shown in a loop on the root window, after the last micon frame is shown, the animation returns to the first.

Sound files

The notion of the final form of a data structure forming its on-screen iconic representation causes greater problems of distraction when considering sound files, than may arise from the movement of a micon at the edge of the user's visual field. If a sound file is looped and made audible whenever the attached visual icon is visible on-screen, as the number of audible sound files increases it will become harder for the user to distinguish the file sought from the background noise. The confusion of sound generated will also tend to annoy other users nearby. While users are able to distinguish the sound sought from a small number of simultaneous background sounds, an ability relied upon in the auditory browser system of Fernström and Bannon (1997), and while simultaneous sounds can be used to draw the user's attention to malfunctioning devices that are not visible, but which are audible (Gaver, Smith, and O'Shea, 1991), determining which icon acts as the *handle* for the sound recognised or wanted poses a considerable problem. A number of solutions to this problem can be found. An example is Kobayashi and Schmandt's (1997) Dynamic Soundscape which maps sound into a loop in an auditory field around the user's head, a speaker is heard moving around the loop with the speaker's topics being positioned in certain arc segments of the loop. The user may then, via a touch pad, indicate a particular topic to be replayed, or may jump around the loop, by indicating the position of the audio segment they wish replayed. This system, however, only allows a single audio file, albeit many segments of which, to be replayed and accessed. Other systems which have multiple, different, sound sources playing from fixed positions in the audio space (for example, Schmandt and Mullins, 1995) are limited in the number of sounds that can be located and differentiated. Alternatively they only play structured sounds which allow attention to be shifted to, or attention to be drawn to, a different sound source. Rather than have multiple sound sources playing at the same time (which would be the approach for sound files

following the notion of self-representation), with its obvious difficulties, a better solution is to easily permit self-representation to be brought about, rather than make it the default behaviour of the model world. The viewing cone, introduced by Mander et al. (1992), permits selected emphasis display of files. When activated over a file icon, a cone expands to reveal more of a file's contents and to provide sufficient additional views to further aid unique identification of the file, not just determination of the file's category. By selecting the particular emphasis to show sound within the cone, as the cursor passes over sound files, the cone appears and the sound, the data making up which is stored in the file, is heard. To be consistent with the interaction style of Medusa, the toolbar, which is discussed further below, associated with categories of sound files includes the methods **Play** and **Stop playing**.

Hypertexts

The application program Hypercard uses the icon shown in Figure 7.3 to denote a Hypercard stack. This clearly reflects the metaphor adopted in a number of hypertext systems, that each node in the graph is a card with a piece of text written on it. There are claims, though, that the Hypercard system is a compromise forced on its designers following legal action taken by the inventor of a system termed *Zoomracks* which is based on a "card and rack" metaphor (Heckel, 1996). The Hypercard icon, and metaphor, provides no notion of the links that connect buttons, be they icons on a card or short strings of text, to other cards in the stack. Hypertext systems will often provide an overview of the entire hypertext which renders the entire graph to ease navigation and determination of the user's current location, such an overview could form the icon generated to depict a particular hypertext. Again, such icons are unlikely to be sufficient to uniquely identify the hypertext, and if the graph is too large and the connections are too numerous to render without intolerable aliasing, a

standard icon denoting the category of on-screen object to which the file belongs might need to be employed.

Figure 7.3 A Hypercard stack

HTML source files

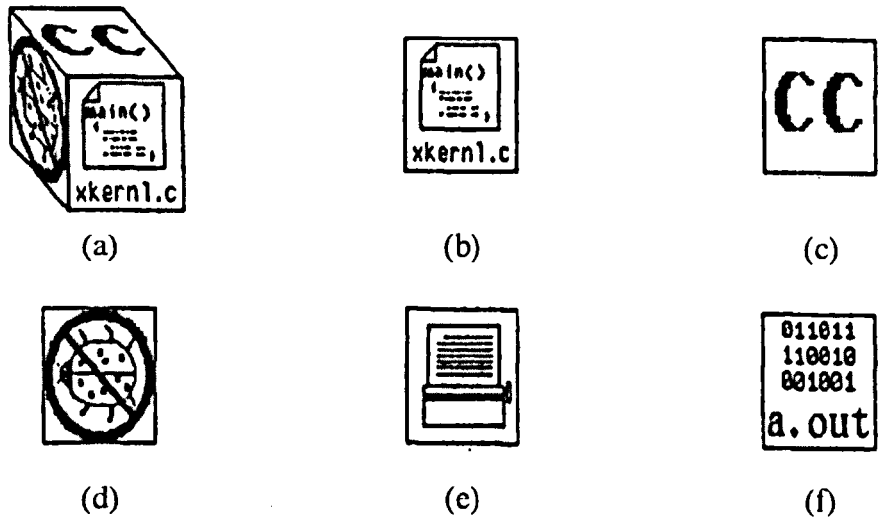
HTML files, are an interesting exception to the class of hypertext files. Unlike many hypertexts, all the information and media fragments in the graph are unlikely to reside in the same filespace. A rendering of the entire graph is therefore impossible to construct in a way that would form a meaningful icon, again a compromise would be to adopt an icon denoting category membership. HTML files present a problem when interpreting the **Open...** command selected typically from a menu bar, or double clicking on the icon. Usually, as mentioned above, the application denoted by the icon, or an application associated with the icon using a software tool is run and the file loaded for the application to process or display. HTML files present the difficulty that while they are usually employed as documents to be displayed by a world-wide web browser, they are also computer programs which are interpreted by the browser to produce a rendering of a particular node in a hypermedia graph. Depending on the user's current tasks, they may wish to edit the HTML program using a text editor tool, or display it using a browser. Different action sequences, or methods, must therefore be familiar to the user, whereas a single interface mechanism which makes the membership of different categories of the file apparent to the user could be employed to overcome this difficulty. We shall discuss such a mechanism further below.

Discussion: Self-Representing Icons Versus Other Icon Designs

Traditionally icons have been static representations of data structures, whether or not the underlying data structure is static or dynamic, in our distinction. Where appropriate, some self-representing icons use animation. According to Baecker, Small and Mander (1991: 1) animation in the user interface helps the user to answer the questions "what is this?", "where have I come from and gone to?", "where am I?", "what can I do now?", "what can I do with this?", "how do I do this?", "what is happening?", "what have I done?", "why did that happen?", and "what should I do now?" Icons that denote files typically help answer the question "What is this?" Self-representing icons are intended to be *richer* (Houde and Salomon, 1993) than the simple class membership denotations criticised above, but are also intended not to invoke the unwanted concepts that ecological icons will, despite their ability to ease location tasks (Ark et al., 1998). Also, while self-representing icons may not prove to be the simplest icon form, the lengthened search time for more complex icon designs demonstrated in Byrne's (1993) study only becomes pronounced in sets where the icon wanted is one of 12 or more displayed. Byrne himself admits that visual search is not the only task performed on icon sets, and it is these other tasks that must also be supported by an interface.

Animation in the user interface may also help answer the question "what can I do with this?" An interesting method of addressing this question is Henry and Hudson's (1990) multidimensional icon. In a multidimensional icon, shown in Figure 7.4, icons depicting different views of a file are texture-mapped to the faces of a cube. Combinations of mouse movements and mouse button clicks allow the cube to be rotated so that a different icon lies parallel to the plane of the screen. This icon then may be selected. The drawback of a multidimensional icon is that rather than just referring to the affordances of the file, it also refers to other distinct objects. The execution view of a C language file, shown in Figure 7.4(f), is a reference to a call to

execute a compiled object. This compiled object is a file distinct from the source file, and should compilation of the source file fail the execution will fail, hence semantic errors are still permitted by this approach. The means by which file affordances are treated in the Medusa system are described below.



(a) (b) (c)

(d) (e) (f)

(a) The multidimensional icon.

(b) - (f) Faces of the multidimensional icon cube shown in (a).

Figure 7.4 A multidimensional icon denoting a C language file (Henry and Hudson, 1990: 134).

7.2.3 What Are Files?

We have sketched above how the contents of files may be depicted employing a method termed "self-representation" to give an on-screen depiction intended to aid location and identification of files. An issue that must be addressed by a product-oriented view of metaphor is the problem of finding real-world counterparts to all types of notions employed in filing systems. Within the DESKTOP metaphor, some icons denote files and folders denote directories, but this is an example of a metaphor that breaks down. In the Unix filing system, to which a number of graphical user interfaces based on the DESKTOP metaphor have previously been developed,

directories are merely files that contain a list of data structures associating a text string, the file's name, to an *inode*. Inodes are special data types that index a number of blocks of data in the physical file space. Within a particular directory, what is listed as a file is just an instance of a text name being associated with an inode, files are not listed, instead these *links* are listed. This has consequences for the semantics of tasks performed in the model world. For example, while a user may think they are deleting a file, if another user entitled to use the system has a link (alias) to the file in one of their directories, the file itself will not be deleted, it will just be invisible to the user that deleted it. In the Apple Macintosh system and in Microsoft Windows 95, by contrast, aliases may be created inside folders which are links to files in other folders, opening an alias will have the same effect as opening the original file. The alias may be deleted without causing the original file to be deleted, but if the original file is deleted, opening an alias will cause an error message to be displayed, as the file that the alias links to no longer being present. A physical metaphor simplifies the notion of deleting a file, but as files may be aliases and not files, other tasks, such as opening files, cannot be accounted for as easily in a physical metaphor.

Vahalia (1996: 220) suggests that "the *file* abstraction acts as a container for data, and the *file system* allows user to organise, manipulate, and access different files." The notion of files as a container is subject, however, to problems in addition to the notion of a file as a single physical object. When copying a file into a folder, for example, on a volume on which there is insufficient physical space, one method to perform this task might be to employ an application package which can split the file into a number of pieces which are each small enough to be stored on small volumes such as floppy disks.. Another method might be to employ an application that can encode and compress the data in the file. Such tasks are more readily suggested by notions other than files being thought of as containers. The traditional Unix notion of files being a sequence of bytes suggests these tasks more readily, for example. Copying a file, if files are containers, will require duplicating the container, or

having a container for the duplicated data to be placed into. The container metaphor for files is again an example of a metaphor that breaks down quickly.

7.2.4 An Ontology of Invisible Objects?

Another issue that creates problems for the use of metaphors in supporting tasks that alter the state of file systems is the use in some operating systems of *hidden files*. These are links, files, or directory entries that exist within a file space but which remain hidden from the user unless a special task is performed to make their iconic representations visible, or to reveal their name in a directory listing. Until an object becomes visible, or can be named, it cannot be acted upon. Therefore in Smith's (1987) terms, a magical feature is required of the user interface in order to make the hidden visible. In a system that attempts to implement a physical world metaphor to account for a file system that supports hidden objects, the metaphor must fail, and systems prototypically described as implementing the desktop metaphor are notable for not supporting hidden files in the file system. In systems where hidden files are allowed, such systems are usually those where a graphical user interface is imposed upon a file system and existing disk operating system, interaction with which has previously been conducted using a command language. The need to support the magical feature needed to make hidden objects visible has consequences, as will be discussed below, for how groups of files are depicted, and on how directories are represented.

7.2.5 Numbers of Objects - Directories and Containers

As discussed above, file systems are made up of files and directories, which are stored on physical volumes (fixed or removable disks). In the DESKTOP metaphor, directories are depicted by folders, which may contain a number of files or

documents. Unlike physical folders, however, they may also contain folders, and so forth until the limit on regression determined by the maximum size of file names (normally a hard system-imposed limit) is reached. Again, the folder metaphor is an example of a metaphor that breaks down quickly. The view adopted in the QPT models presented in Chapter 5 is to claim that directories and disks, but not files, are containers. The container has advantages as a metaphor for directories in filing systems (though, as seen above they are less successful as an account of files), containers may be placed inside containers without the metaphor breaking. Containers have a capacity, so the user can know whether an attempt to move a file into a container will be successful. Unfortunately the capacity of a container can be hard to determine, or may not be fixed. Volumes, such as floppy and hard disks, have a capacity, namely the number of bytes available for the storage of data blocks. The capacity of a directory is not fixed, however, instead it is the result of the constraint equation shown in Figure 7.5.

$$potential\ free\ capacity\ of\ a\ directory = capacity\ of\ volume - \sum_{i=1}^{t} size\_of\_file(i)$$

where t = the total number of files in the file system on the volume

Figure 7.5 The potential capacity of a directory

In many graphical user interfaces, if the user attempts to copy a file into a container which has insufficient capacity the system might allow the user to perform the actions that activate the copying process (clicking on the icon, moving the mouse until the pointer is over the destination folder's icon, then releasing the mouse button) but it will display an error message. Again, the physical world metaphor breaks down and the layers of description separating the model world from its underlying implementation, and the different processes at work in the model world and in the underlying implementation, as described in Chapter 5, become apparent to

the user. It is such system behaviour that underlies the claim that in direct manipulation interfaces, the user cannot make *syntax errors* in their dialogue structures, but they can make *semantic errors.*

We are required to provide a means of interacting with the file system of a computing system to which Medusa is intended to be a graphical user interface. Two options exist that fit the aims of the Medusa system and the notion of making visible those aspects of the underlying system necessary for understanding the system. One is the existing, popular, desktop, user interface, where opening a folder icon reveals a window containing icons depicting the files contained in the directory that the folder denotes. Such a system supports the magical features needed to list potentially useful information about the files, their size, creation date, to list file names in alphabetical order so as to aid location of a file sought, and to make hidden files visible. The difficulties remain of accounting for such a solution within a basic physical world metaphor, accounting for the mismatch between the representation and the underlying data structures, and deciding how the issue of positioning icons within the window should be managed, however. Such functionality reminds us of the original meaning of an icon as depicting a closed window, and suggests that far from being physical objects in the electronic world, the folder is a window onto an application for managing files.

As icons in some filing systems depict links, (entries in a data structure associating text names with the actual file's location on the disk), opening a folder to reveal its contents is simply to read the file and list the entries in this data structure. Reading a file, the contents of which are links to files stored elsewhere in the file space, some of these files also contain a sequence of entries in a directory, is a task performed by a browser application such as Netscape Navigator. Thus attempts by some operating systems manufacturers to integrate browser applications more closely with the file management system are thus justified by the need to support file management tasks,

legal arguments not withstanding. One can see that as the directory file (that contains links to files in the file space) becomes more sophisticated in the way that links are rendered (using graphics and video fragments), and as the layout of the directory file's contents when rendered within a browser window becomes more flexible (perhaps user layout of links in the browser is allowed), a browser becomes indistinguishable from the folder graphical interface. Except that, in the case of the browser metaphor, fewer breakdowns between the user interface and the underlying data structures occur.

Our first attempt at providing a user interface to a file management system then is a browser application run by opening a directory file. The design of suitable icons for these files remains to be determined, but these too can be accounted for using the notion of self-representation. The problem of file movement around the file space must also be addressed, the problems caused by the use of a physical world metaphor have been documented above, we postpone presentation of a solution to this problem until Section 7.2.7.

7.2.6 The Computer-Computer Metaphor

The use of a metaphor in the design of the first Medusa system that is wide in scope is made according to the idea of the computer being a metaphor for computing systems (Treglown and O'Shea, 1993). The notion of making the user aware of relevant aspects of the system necessary for understanding applies not only to file representations. Where information about the status of processor and memory usage, data structures common to applications such as input buffers, and devices connected to the workstation, is also required, it is provided on-screen. In (Treglown and O'Shea, 1993) it is proposed that a graphic such as that shown in Figure 7.6 appear on the root window. This may be examined using common interaction tasks to reveal the state of the device to explain its behaviour (for example, processor load and a list

of runnable processes), and the state of some parts of the system may be altered (current active process, destination printer, and so on). As this graphic does not allow the user to examine all the relevant parts of the system, as in Myer's (1988) Peridot visual programming system where it is necessary to be able to interrogate and alter the state of devices such as the mouse and pointer (attempting to taste one own's tongue is an analogy[3] of the task to be performed in the model world), the decision to place the graphic in Figure 7.7 on the root window is now adopted.

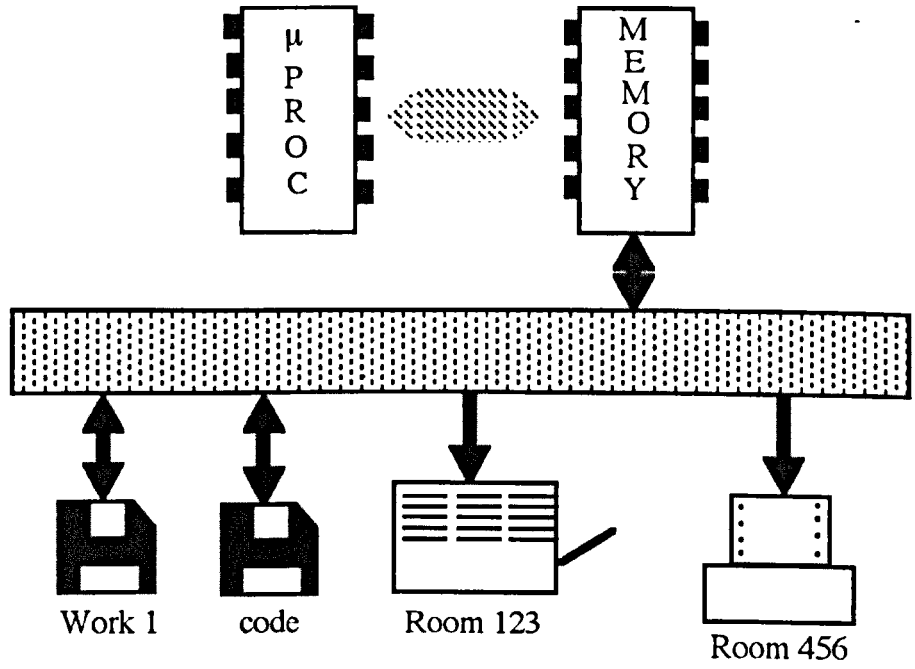


Figure 7.6 The first design of device description in Medusa

[3] Attributed to the playwright and actor Ken Campbell in the Channel 4 television series "Brainspotting".

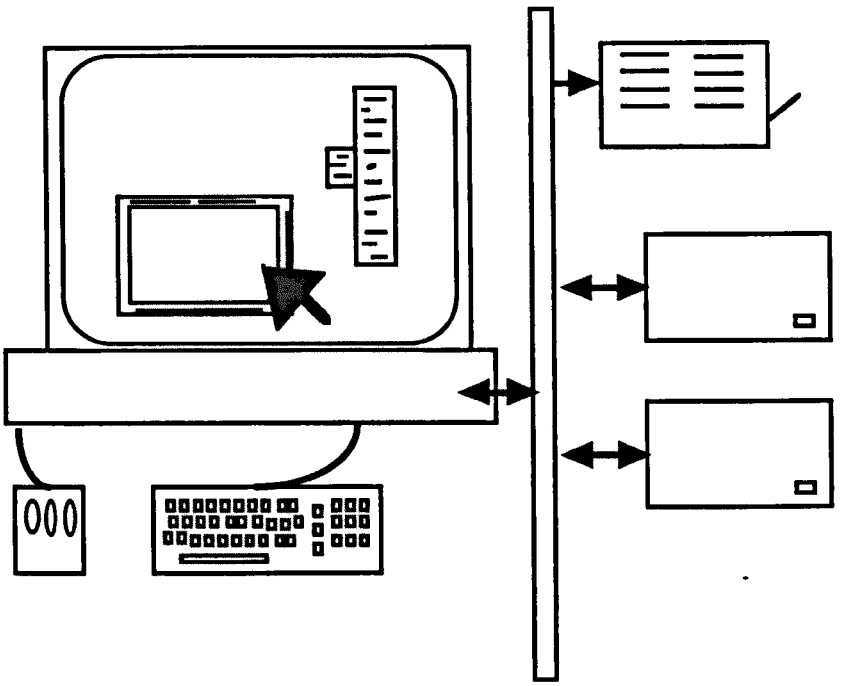


Figure 7.7 The second design of device description in Medusa

The graphic shown in Figure 7.7 allows the user to modify the presentation and behaviour of the Medusa system. Normally knowledge of preferences or control panel applications is required to modify the sorts of values that the Medusa system allows access to via the same interaction style as the rest of the system.

7.2.7 Performing Tasks in Medusa

While metaphor-based graphical user interfaces are termed direct manipulation, the conversation metaphor, whereby users are said to have a conversation with some unseen agent about a (normally unseen) task domain, also plays a part in interaction. While some variables can be directly manipulated, the issuing of commands to on-screen objects by selecting commands from a menu bar common to many systems is based on an subject-object-verb syntax. The object of interest is made current, and the action to be performed on it is selected from a tool icon or menu bar. The idea of a current object introduces additional concepts that the user must be aware of if they

are to be able to conduct even basic interaction with a system. Phillips and Apperley (1991: 14) say that:

> "The Macintosh interface is based upon the desktop metaphor. The Finder manages objects hierarchically in the form of applications, documents, folders and disks on the desktop. 'Closed' objects, which are represented by icons can be selected (made current), and once current can be opened, moved, discarded, etc. The contents of 'open' objects are displayed in windows, which can be selected (made active), and moved, sized, scrolled, etc... The interface is based on a single object-action model — that is, at any time there is a single current object or group of objects on which a specified action is carried out."

The study reported by Carroll and Mazur (1986), discussed in Chapter 4, found that even a task as basic as highlighting (making objects current) proved difficult for users. Analysis of the failings of many user interfaces that derive from the desktop metaphor has apparently found that the notion of the current application causes user problems. Where a number of overlapping open windows appear on-screen, each being employed by a separate application program, windows being obscured when a single window is made current reportedly causes users some confusion (Halfhill, 1997). The problem of user input (from the mouse and keyboard) being directed by the window manager to application windows other than the one expected, the issue of *focus*, is already a well-known problem.

Basic interaction tasks are sometimes described in terms of the input devices that can be used to support them, or in terms of virtual input devices, those that generate the types of input required. Phillips and Apperley (1991: 11) summarise basic interaction tasks, as shown in Table 7.1.

Task	User Action	Equivalent Virtual Device
Position	Indicate a *position* on the display	Locator
Orient	*Orient* an entity in 2D or 3D space	Locator
Quantify	Specify a value to *quantify* a measure	Valuator
Select	*Select* from a set of alternatives	Button, Pick
Text	Input *text*	Keyboard
Path	Generate a *path* (series of positions) over time	

Table 7.1 Basic interaction tasks and virtual devices

In their analysis of the Macintosh system, in particular the Finder, Phillips and Apperley find that all interaction tasks, including the complex dragging tasks that the user must understand and perform in order to create groups of icons, can be reduced to selection tasks. The generic *get value* task is an example that reduces to selection tasks, as shown in Figure 7.8.

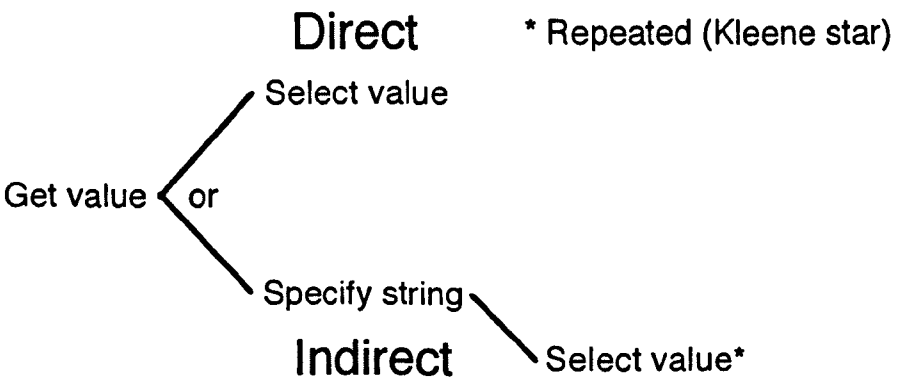


Figure 7.8 Get-value sub-task (Phillips and Apperley, 1991; Page 15).

Following Phillips and Apperley's analysis of interaction tasks, and following users' difficulties with the basic notion of current objects and windows, the approach to

interaction adopted in the first version of Medusa is an *object-message* style of interaction. Instead of making the object on which an action should be performed current, termed *existential reference* by Lakoff (1987), *deitic* reference is preferred, where reference is made to an object and commands are selected from a toolbar that appears alongside the object once it is clicked on. This toolbar is described in more detail below.

A number of solutions to the problem of selecting commands to be applied to an on-screen object exist. As described in Chapter 2, in ARK *named* buttons, that denote operations, are dropped onto the object to which the operation should be applied. This solution however still permits semantic errors to be made, where operations are applied, or messages sent, to objects that cannot respond in any meaningful way (the method is not part of the object's interface). In such a case the button falls through the object, an action that is very magical and hard to account for. An interesting alternative is Muller's (1988) multifunctional cursor. In the multifunctional cursor operations are loaded into slots in the cursor and applied to objects by traditional mouse button clicks, an unloaded cursor is shown in Figure 6.9. The multifunctional cursor has drawbacks, however. For example, the number of operations that can be loaded into the cursor slots is limited to twice the number of physical mouse buttons available, neither are semantic errors prevented. In addition, while some appeal to stimulus-response compatibility can be made in mapping cursor slots to actions, the icons in the slots must be interpreted in order to determine which actions will be performed. This task is made more difficult for the users of Muller's design by them having to employ the typical homonyms, abstract symbols, and puns found in UNIX icon sets, such icons being among the poorest scoring in icon recognition tests (for example, Rogers, 1986). Also of interest is the tool tray from which operations are loaded into slots in the multifunctional cursor, this is a rectangular array of icons denoting the operations that can be loaded into the cursor. Muller, however, does not address how the tool tray may be retained close to hand in multi-screen or

collaborative systems as the cursor migrates around a model world that may be larger than a single screen in size.

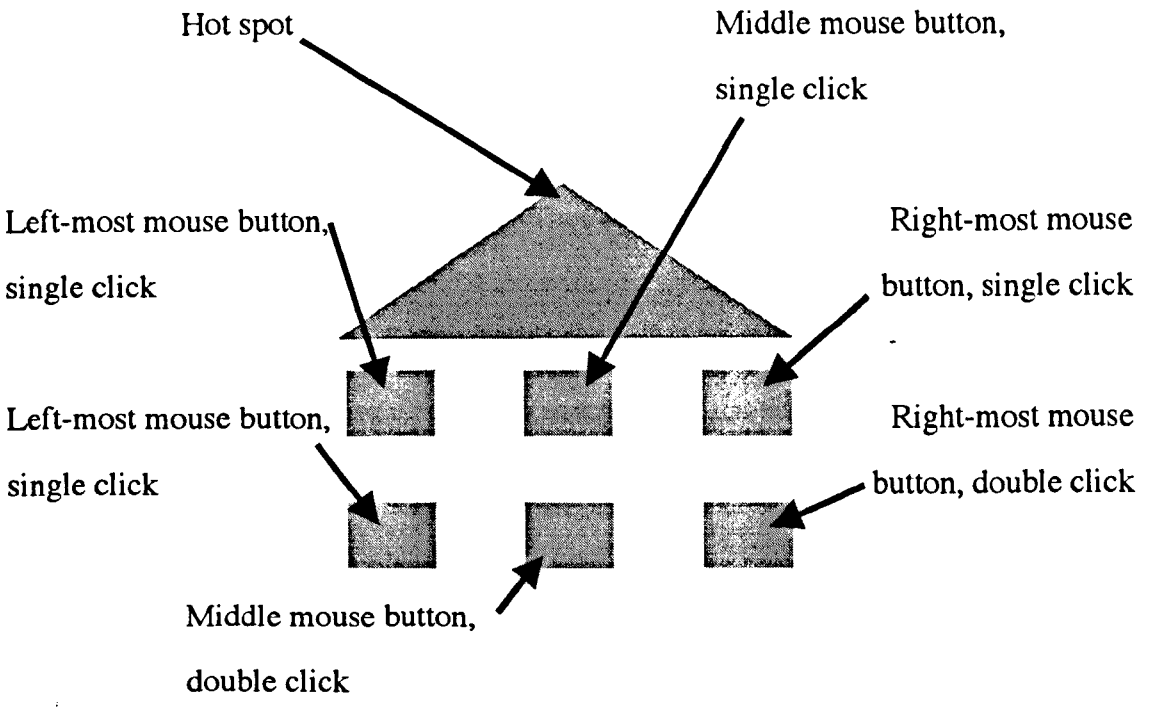


Figure 7.9 An unloaded multifunction cursor for a 3-button mouse

Most desktop metaphor systems, though not the Xerox Star, place a menu bar either at the top, bottom, or the side of the screen. The menu bar is not adopted in Medusa, it being required in systems that employ the active or current object notion, which Medusa does not. All options are thus selected from the toolbar, shown in Figure 7.10, a design that allows two-handed input (Bier et al., 1994) to be supported, if the user so wishes. Clicking on an object of interest causes a toolbar listing the operations that may be applied to the object to appear, from which an option may be selected. The list that appears in the toolbar depends on the affordances of the object and the operations that may be performed on instances of the class or category that the object belongs to. The list also depends on the application programs installed in the system that might be able to run using the object as data; and on the object's current state. In this way, semantic errors cannot arise as operations that are not

meaningful in the current system state are not offered to the user. The toolbar also overcomes the problem of having operations close at hand. The toolbar combined with the computer-computer metaphor, achieves much of the clustering of operations with their associated underlying computing functionality reported by Tullis (1985), but in a model world direct manipulation interface. The use of the object-message interaction style, and the use of micons and continual state feedback, combined with the use of the computer-computer metaphor for making the underlying system visible has similarities to, but was developed independently of, Maloney and Smith's (1995) Morphic system.

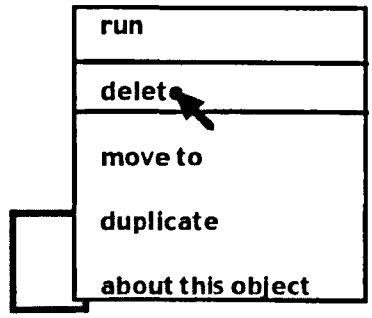


Figure 7.10 A sample toolbar

7.2.8 Groups of Objects

In order to reduce the time taken to perform simple tasks, a number of desktop-based user interfaces allow files to be grouped so that a single command may be issued to all files in the group at the same time. Making a number of icons current requires that the user performs a repeated <shift>+click action sequence, or drags a bounding box around the icons to be made current (lassos them). Phillips and Apperley (1991) show that the mouse-dragging action required to lasso icons also reduces to a selection task. To allow groups to be built, the Medusa toolbar provides **Add to group** and **Remove from group** commands among others. Groups will be treated differently from individual files, and the toolbar will list only those

operations that can be applied to the group as a whole if the object clicked on is currently a member of a group.

7.2.9 System Feedback

After the user issues a command to any interactive system, according to Norman's (1984) model, they must first evaluate the system response in terms of their goals and then either issue commands that correct unexpected system responses, or they must issue further commands that take the user closer toward completion of their goals. Much of the motivation for the Medusa system is concerned with the need to address the fact that system feedback is often not immediate, and that the computation performed to complete a user-initiated action has an often perceivable temporal duration. As interactive systems are not implemented on infinitely fast hardware (Dix, 1987), the need to manage the flow of interaction in a system subject to delays, lags, and seemingly lengthy computation must be addressed.

Buffers

Where computation is lengthy, appropriate progress indication is required (Myers, 1985). Like providing UNDO facilities, as will be discussed below, providing progress indication requires depicting seemingly intangible properties of the system such as the amount of computation performed, or the current state of an event queue. These attributes of the system have no corresponding concepts in the task domains that the systems described in Chapter 2 provide interfaces to. In the case of buffers, Dix (1991) provides a design solution in keeping with the ideas underlying Medusa. Dix's "munchman" buffer depicts keystrokes placed onto the event queue as a result of the user typing while the target application is too busy to process them. When the target application processes an event, the corresponding character is removed from the depiction of the buffer's contents. This on-screen object is not *equal opportunity*

(Runciman and Thimbleby, 1986) the user may only place items onto the buffer (including the delete character), the application is the only party that may remove items from the buffer.

Progress Indication

While user interfaces are event-driven systems, and the interface programmer must be concerned with giving semantics to events in terms of how the display and underlying software changes, user interfaces and interaction with systems, according to Dix and Abowd (1995) are also concerned with *status*. Unlike events, which for conceptual and mathematical convenience are assumed to be instantaneous, *status* describes aspects of an interface that have a constantly available value. Some events do not, or need not, change mappings between the status of the user interface and the status of the underlying software, some events, however, do and feedback is required as a status-status mapping is restored.

In Medusa, such restorations of status-status mappings, and progress indication, are required if the user initiates file moving and copying operations. Copying, for example, implements semantics similar to the following (taken from Stevens, 1992: 56):

```
int main(void){
     int  n;
     char buf[BUFSIZE];
     while ( (n=read(STDIN_FILENO, buf, BUFFSIZE)) > 0)
          if (write(STDOUT_FILENO, buf, n) != n)
               err_sys("write error");
     if (n < 0)
          err_sys("read error");
     exit(0);
}
```

A metaphor for the copying operation, based on the notion of visibility of relevant (otherwise hidden) system components, will therefore depict the buffer filling and

emptying. The effects on the file copied and the copy produced depend on depictions of the semantics of the read and write operating system calls that have yet to be fully resolved. The Medusa system's depiction of this semantics, it should be noted, differs from Dourish and Button's (1998) *reflective* account of the same operation. Reflection is described in more detail in Chapter 9.

Mouse-ahead

Typed characters may be placed in a buffer until the target application is able to process them, this being termed *type-ahead.* It is also possible for events generated by use of the mouse to be queued until they can be processed, this being termed *mouse-ahead.* Some treatments of mouse-ahead, for example by Dix (1991), limit the number of mouse events, in particular mouse clicks, to that required to complete a semantically meaningful subtask. If a larger event queue is maintained it is possible for semantic errors to occur once the queue is processed, events meaningful in a busy, frozen, model world are unlikely to be meaningful as objects are altered and move when events are eventually processed. Pérez-Quiñones and Sibert (1996) assume that direct manipulation interfaces must include a conversational component in dialogue, direct manipulation being a collaborative process between the user and the event processing system. If events cannot be processed in the current state then the user must be informed, so as to allow them to alter their behaviour if required.

Pérez-Quiñones and Sibert's model has five "simulated states of understanding" (SSOU): *ready*, *processing*, *reporting*, *busy-no-response*, and *busy-delayed-response.* These states are intended to reflect conversational behaviours in dialogues between people and in speech recognition systems. These states and transitions between them are shown in the Statechart-like system in Figure 7.11. In the internal loop, feedback denoting that the system is in the processing state can be omitted if processing can be completed in a short enough time that breakdowns in the system

behaviour do not occur. Transitions to busy states occur if processing will take, or is taking, some time. In the internal loop states are normally depicted by the conventional icon shape (pointer or hand), to denote busy states an alternative icon is required, Pérez-Quiñones and Sibert use a stop sign icon to denote busy-no-response and an hour glass icon to denote busy-delayed-response. In the Medusa system, because of the system architecture adopted, which is discussed below, we do not regard the system as a whole as being in a particular state, only that certain classes of object in the model world may be in a particular state. Thus a version of Pérez-Quiñones and Sibert's SSOU model will be built into interaction with *each* on-screen object. Further work will examine alternative icon designs to denote busy states, while the hour glass is a reasonable metaphor for a busy-delayed-response state (although we *can* interrupt the task of boiling an egg, for example), the stop sign is a poorer metaphor for the state it denotes.

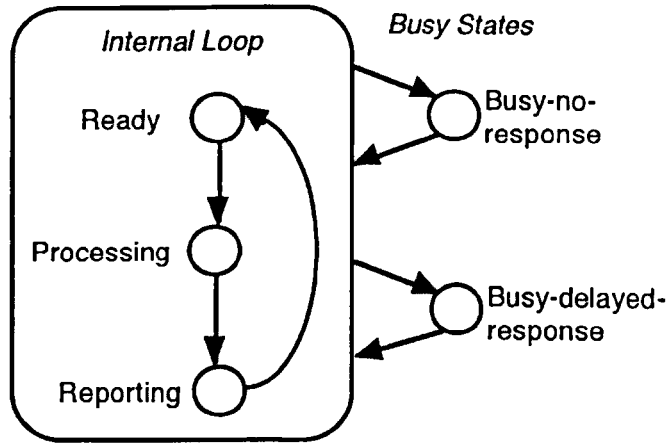


Figure 7.11 SSOU feedback states

(Pérez-Quiñones and Sibert, 1996: 318).

7.2.10 Help

The study of first-time users of the Macintosh, reported in Chapter 3, found that while on-line help facilities were used, the help they provided was limited and context-independent. It was proposed that help should, nevertheless, be provided and

should be object-based. When help about an on-screen object is requested, information about the object, its current state, and the states it may enter by issuing of one of the currently applicable operations is provided. Help is an option available on the toolbar, its purpose is to give the user a semantics for commands applied to on-screen objects. Object-based help addresses the dialogue determination problem, where options available to the user are detailed, but choice is simplified (Kirsh, 1996). Help facilities are also used to detail the history of an object in order to provide information which allows the user to uniquely identify the object pointed to by the icon link. Help assists the user where the icon's design alone does not meet this requirement, and it permits the user to add whatever comments prove useful in aiding them and other users to determine the data contained in the file. Filenames, as a result of the process by which they are developed, tend to be meaningful only to the user who named the object. Medusa will also allow how the help system behaves to be modified, it being self-representing, the user should have access to, and the ability to modify, the help system.

7.2.11 The File Manager

Interaction with the underlying file system using the Medusa user interface has been partly detailed above. The depiction of directory listings was discussed, but the issue of how files are moved around the file system was not determined. The QPT-based analysis described in Chapter 5 showed that the processes that affect objects in the on-screen model world differ from those conjectured to affect objects in the underlying software. It was shown that tasks as seemingly simple as moving files by dragging them are subject to a considerable mismatch, the physical world metaphor cannot account for the behaviour of the system image. To make the process affecting the data structures apparent, and following the task analysis conducted by Phillips and Apperley (1991), moving a file around the file system requires that the user point to a file or group of files and select the **Move** command from the tool bar.

The user must then specify a destination by pointing to a directory listings file, or within its open window. Following the structure of the QPT processes given in Chapter 5 and Appendix A, a path must exist between the volume on which the source files are stored and the destination volume. The on-screen graphic shown in Figure 7.7 permits the user to establish this path, or to check that a path is still in place.

File deletion, at least in UNIX, is the process of removing a link from a directory entry listing file and freeing the list of blocks that the file uses for use by other, new, files. The data is not destroyed until the data blocks are reused. The Medusa system does not employ complex physical world metaphors such as the trashcan, or the black hole employed in some ARK simulations. Instead, deletion is just another option available on the toolbar of messages that can be sent to an object, in a way similar to dropping a physical delete button onto an ARK on-screen object removes the object from the model world. As directory entries are only links to files, the actual file persists until no more links to it exist. We address the possibility of undeleting files in Chapter 10.

7.3 Implementing Medusa

The first Medusa system has been described in some detail above, and while a usability analysis of the design is undertaken in the following chapter, a full appreciation of the pragmatics of interaction with a Medusa system would only be gained from a full implementation of the system. A secondary aim of the research programme begun and reported in this thesis is to understand more fully the nature and limitations of direct manipulation. We also seek to apply formal methods and models during the development of user interface features so that the intended behaviour of these features is known, verifiable and hopefully guaranteed. An implementation of version one of the Medusa system was begun, as was ongoing

work in the specification and refinement of interactive systems based on an object-oriented formal notation. In this section, we describe below the work undertaken in this area as applied to the design and implementation of a Medusa system.

7.3.1 Use of the Agent Notation and Language

A trend in current user interface design views the system as being made up of a number of small, possibly interacting, modules. Approaches that follow this trend include PAC (Program, Abstraction, and Control) agents (Coutaz, 1987), and the MVC (Model, View, Control) paradigm of the Smalltalk-80 programming environment. The model adopted in the design of the Medusa systems is the agent model and notation (Abowd, 1990). The agent model views systems and their user interfaces as being composed of a number of inter-connected, communicating components. These components, termed *agents*, in turn, consist of three parts (depicted in Figure 7.12). These parts are a persistent internal state which changes as the internal operations are invoked as the agent receives event messages from other agents; a communication part that lists the one-way communication channels that connect agents together and names the event messages that may be sent or received along each channel; and an external behaviour part that defines the sequences of event messages that the agent is prepared to engage in. This last part supports the *interaction design*, the interleaving of user input and system output so that tasks are supported by the system and the behaviour of the system is reasonable and comprehensible to the user. The external behaviour component also distinguishes the agent model from pure object-oriented models in which sequences of method calls are defined by the arrangement of objects into interconnected layers corresponding to the lexical, syntactical and semantic layers of a linguistic approach to user interface management (for example, Sibert, Hurley, and Bleser, 1986).

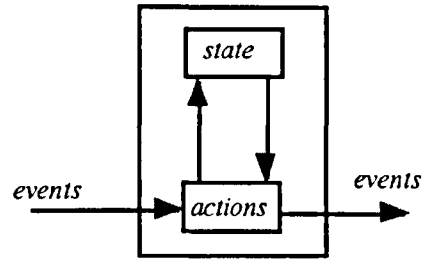


Figure 7.12 An agent

7.3.2 System Architecture

The first version of the Medusa system, the one that has received most design effort, adopts the UMA architecture introduced by Took (1990a, 1990b) which is shown in Figure 7.13. Took (1990a), in addition to providing the architecture adopted, provides arguments as to why a user interface architecture should be adopted at all in preference to existing user interface services such as toolkits and user interface management systems. The most compelling arguments for adopting a user interface architecture, in terms of implementations of Medusa, are the drawbacks, paraphrased from (Took, 1990a), listed in Table 7.2. In the UMA architecture, an interactive system is composed of three parts, the *Application component* contains the functionality of the system and receives commands from the *User component* and may send commands to the *display Medium.* The User component receives events generated by the user via input devices such as the mouse and keyboard and either forwards them to the Application or interprets them as commands to be issued directly to the display Medium. The display Medium, is a passive component, it serves only to maintain a display model, or implement a display operating system, which is altered by the component acting on commands received from the User and Application components, and rendering the display model to produce the screen contents. Adoption of a Medium also does not rule out future consideration of a collaborative version of the Medusa system, various present functions allow support for multiple displays, the implementation details, and appropriate screen sharing approaches (for example switchable workspaces, rooms, or Kansas-like

environments) have yet to be fully explored, however. The UMA architecture is depicted in Figure 7.13.

Reasons for rejecting window managers

- They only provide an incomplete data abstraction,
- Applications are given no abstraction for the contents of windows.
- All that is provided by a window manager is a set of low-level graphics primitives, or possibly also a confusing hierarchy of panes, panels and sub-windows.

Reasons for rejecting user interface management systems

- The need to support interleaved dialogues, spatially-multiplexed tasks over different windows and applications, which are hard to model syntactically.
- Semantic feedback, where engagement between on-screen objects and underlying semantic objects is expected, is hard to support if dialogue management is separated into a distinct component.
- Dialogue abstraction is more suited to procedural applications, but not where the user has freedom of action.

Reasons for rejecting user interface toolkits

- Design by modification, new classes of object are hard to create, it is easier to modify an existing class. Designs are limited by the quality of the set of basic components.
- Poor separation and high semantic seepage, no clear dividing line between application code and interface code.
- Objects handle their own interaction, optimal updates and screen synchronisations require additional global superstructures.

Table 7.2 Drawbacks of user interface services and reasons for adopting a user interface architecture.

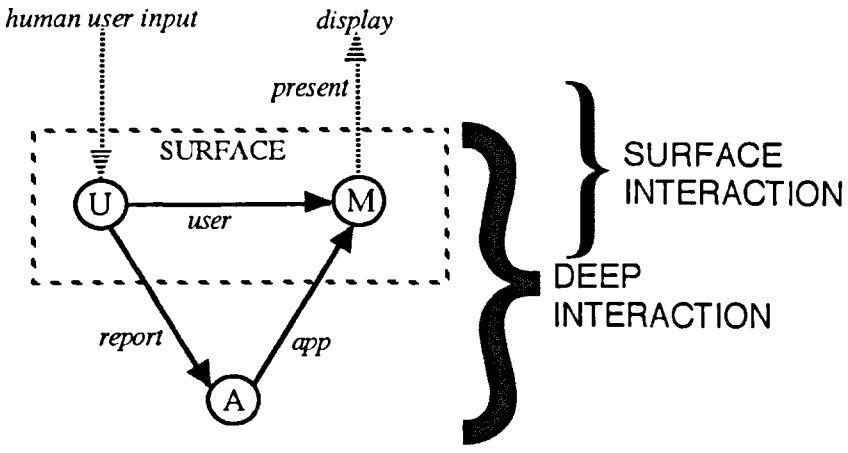


Figure 7.13 The UMA user interface architecture

The UMA architecture introduces the notion of *surface interaction.* Where events do not need to involve the Application they are handled by the *surface*, and so the display contents can be altered without semantic seepage, and without the typical program structure where code for the user interface is merged with code for managing the Application functionality. At the same time, the UMA architecture can support semantic feedback where the application semantics may need to directly alter the display contents without having to consult or inform intermediate dialogue manager components as is often found in user interface architectures that adopt a linguistic approach to dialogue management. The UMA architecture is captured by the following three process expressions given in the notation of Communicating Sequential Processes (CSP) which is employed in Abowd's (1990) agent model to define the external behaviour component of each agent.

U = i:I → user!pick(i) → o:REPLY → (user!c:COM → r:REPLY → U □
report!(i,o) → (i', o') → user!c:COM → r:REPLY → U)

M = user?c:COM → r:REPLY → M | app?c:COM → r:REPLY → M

$$A = \text{report?}(i,o) \rightarrow \mu X \bullet (\text{app!c:COM} \rightarrow \text{r:REPLY} \rightarrow X \,\Box\, (i',o') \rightarrow A) \;\Box\; \text{app!c:COM} \rightarrow \text{r:REPLY} \rightarrow A$$

The concepts of surface interaction, and of distinguishing surface interaction from *deep* interaction, where events do require processing by the application, provide a separation of software components that allows clearer discussion of the components and their behaviour that make up some of the domains that our interface metaphors map between. The notions of surface and deep interaction also provide an alternative view to that adopted in (Dourish and Button, 1998) in identifying points of breakdown and failures of scope in metaphorical mappings in direct manipulation tasks, this shall be discussed further in Chapter 9.

7.3.3 The Application

The application component of the system described in this thesis, and in many of the systems described in Chapter 2, is a simple file manager. The file manager provides mechanisms for simple file maintenance, and provides functions for deleting and moving files and creating new directories within the file space. The file manager can be described by a single agent. The persistent state maintained and altered by the agent describes the graph structure of files and directories within the file store. This agent is based on a completion of the partial description of a file store agent given in Abowd (1991). This agent is, in turn, based on the formal specification of the UNIX filing system given by Morgan and Sufrin (1984) and the object-oriented UNIX filing system specification provided by Meira et al. (1994). The Medusa system is not, however, meant to be a direct manipulation interface to the UNIX file system, the usability failings of which have long been documented (Norman, 1981), and to which direct manipulation interfaces have already been constructed, for example by Borg (1990) and by Lundell and Anderson (1995). The functionality of the Common Desktop Environment front panel in Lundell and Anderson's design is subsumed by

the computer-computer metaphor. Medusa prototypes were, though, planned to run with the UNIX operating system and hence the UNIX filing system must be assumed in a specification at the level of detail captured by the Agent notation. The file manager agent, as implemented, merely provides an interface between the file maintenance functions invoked by user actions and the UNIX system, and is, as a result, a very small part of the whole system. The temporal behaviour of the file system agent is considered to be beyond the control of the Medusa system, as a result no temporal information is provided in the definition of the external behaviour component of the agent. Any user interface to the file manager is responsible for representing in a way meaningful to the user and overcoming these delays, where possible.

7.3.4 A Partial Implementation

An implementation of the Medusa system version one was begun, but remains incomplete. Source code is implemented in C and C++ written for the SunOS version 5/Solaris 2.3 dialect of the UNIX operating system running on a Sun SPARCStation. To speed the implementation, the WIMP user interface component of the system was to be implemented using the XView version 2.3 widget set built upon the X windows system release number 5.

The file manager application component of the system has been implemented as a single class, which, with its associated methods, comprises approximately 200 lines of C++. The class methods reflect the operations performed to alter an agent's state in response to receiving a message from agents that make up the user interface component. The application component makes up such a small percentage of the system code by making considerable use of high-level UNIX system calls. Implementation in a single class reflects that only a single agent is sufficient to specify the Application component. Much of the agent's state-changing operations

were derived from the specifications given by Morgan and Sufrin (1984), therefore the Application component is said to be a design specification, as it was known beforehand that a UNIX platform would be the target for a prototype implementation.

In developing the graphical user interface, a far harder task was confronted. Bass and Coutaz (1991) discover, while attempting to refine a system specified using PAC agents (Coutaz, 1987) to the C language library interface (Xt) to the X window system, that refinement can only be progressed a certain number of steps before the interaction style and code structure imposed by the window toolkit employed restricts the subsequent design choices that can be made. This problem was encountered in the partial development of a Medusa system, and in other applications of Abowd's agents in system design and development (Treglown, 1998). Principled and eventually, it is planned, automated methods for converting an agent-based specification into code are still being developed, and were in a greater state of infancy when an implementation of Medusa was begun.

Approximately 9,000 lines of code of an implementation of the Medusa version one user interface have been developed. This code serves a number of purposes, firstly to provide C++ with an *object* class which the language, unlike truly object-oriented languages such as Smalltalk-80 and Java, lacks. The object class is the most abstract and highest class in the hierarchy onto which the Medusa on-screen object ontology shown in Figure 7.1 is built. The code also begins to implement the UMA architecture on the target system. The code, however, was developed mostly to develop heuristics that would guide the development of semi-formal refinement rules, and eventually type-checking and compilation tools, to be used to help automate the process of generating code from an agent-based specification of the Medusa system. The code developed was also used to make clear the difficulties of attempting to implement on-screen objects that are required to exhibit behaviour that

is not already captured by the widgets provided by existing programming languages and user interface management systems. In addition, we seek to develop heuristics for converting specified agent behaviour into the interaction structure imposed by widgets provided by existing window systems. Simplifying the process of programming the external (observable) behaviour of on-screen objects and widgets, and making it possible to easily modify this external behaviour is a problem that has yet to be solved sufficiently for a satisfactory and complete implementation of the Medusa system to be undertaken. As several man-years' worth of effort was invested in the design of the Xerox Star's icons alone (Bewley et al., 1983), and as the Apple Lisa is said to be the result of 200 man-years of development effort, it is no surprise that a complete implementation of Medusa is not available.

7.4 Conclusions

In this chapter the design rationale of a new user interface design entitled Medusa was described and the criteria and requirements of user interfaces for novices that it is intended to meet were presented. While an implementation of this interface was begun, it remains unfinished and hence full usability testing cannot be undertaken. In the following chapter, we apply usability evaluation and inspection methods that may be employed even when a working prototype is not available and comment on whether the Medusa system meets the requirements placed upon it and overcomes the difficulties of existing metaphor-based systems.

Chapter 8

A Critique of the Medusa System Design

"After we gazed up at the glorious stained-glass windows and exquisite statuary, she led me inside the sanctuary and showed me the ornate carvings on the chairs where the choir customarily sat. At the bottom of one seat was a carving of a dinner scene. My friend told me to stick my hand underneath and feel the hidden surface. Incredibly, the craftsmen who had designed this furniture had actually carved the feet of the festive celebrants under the chairs, even though no one would ever see their loving artistry. They did it for the greater glory of God, because they wanted things exactly right. They wanted to make sure that their work was absolutely flawless.

This briefly reminded me of the decision by the producers of Cannonball Run II to include a cameo appearance by Don Knotts in a film that already showcased Dom DeLuise, Ricardo Montalban, and Jamie Farr, but I quickly realised that this was an inappropriate analogy, and let it go."

— Joe Queenan (1998) *America*, Picador.

8.1 Introduction

The Medusa system design presented in previous chapters is designed to not be subject to many of the failings of existing metaphor-based systems. In order to test this claim, some determination of the usability of the system design must be undertaken. Techniques for examining the usability of a system are classed either as *usability evaluation* techniques or *usability inspection* techniques. Usability evaluation techniques, such as traditional experiments, analysis of system use patterns, questionnaires and interviews, and error rate analysis (Howard and Murray, 1987) all rely on the system being implemented in an executable form, even if just a prototype. As no working prototype of the system exists, conventional usability testing and laboratory-based evaluation methods cannot be employed to judge the usability of the first Medusa system design.

Usability inspection methods, surveyed in Nielsen and Mack (1994), are intended to serve as "low cost" alternatives to usability evaluation methods. Inspection methods can often be performed by evaluators alone, without the need for involving subjects who represent the user population that the system is intended for. Inspection methods also tend not to require a full implementation of the system, they can be performed on specifications and storyboards; and they can often be performed by software engineers who may not be skilled in user interface design. Inspection methods, however, can only find a large minority of the usability faults that laboratory testing can reveal (Desurvive, Kondziela and Atwood, 1992). With no implementation of Medusa available to test, we must employ a suitable inspection method. Below we describe and employ a usability inspection method termed the cognitive walkthrough method in order to examine the usability of the first version of the Medusa system.

8.2 The Cognitive Walkthrough

8.2.1 Interaction and The Cognitive Walkthrough

The cognitive walkthrough is a usability inspection method derived from Polson and Lewis's (1990) CE+ model of learning by exploration. While differing in some aspects, this model can be illustrated with reference to Norman's (1984) seven-stage cycle of interaction, which is shown in Figure 8.1. In CE+, users' goal structures are similar to the hierarchical structures of GOMS. Goals are represented by propositions and are linked to other goals, background knowledge (also represented as propositions), propositions that represent objects seen in the environment and to propositions that denote actions (Polson, Lewis, Rieman and Wharton, 1992). Activation flows from the topmost goal to representations of actions, when an action is sufficiently activated, it is executed. New propositions are created as the environment changes in response to the action performed.

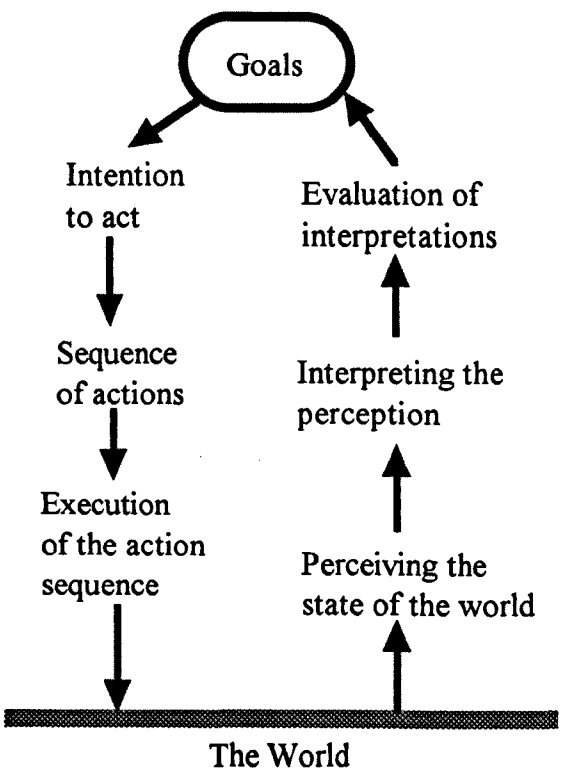


Figure 8.1 Norman's Seven-Stage Model of Interaction

In order for an action to be executed, a chain of associative connections must exist between a goal and an action. "Label following" is one such way in which this chain can exist. In this strategy, used commonly by naïve users, an action, such as pressing a button (using the example given in Polson, Lewis, Rieman and Wharton, 1992), is chosen because the button is labelled in such a way that it shares terms with a currently active user goal. The cognitive walkthrough method seeks to identify points in an interface's design where the chain of associative connections is broken. These points include the label not sharing terms with the active user goal; the link between the label and the button to be pressed being unclear; buttons not being recognisable as buttons; and there being more than one label visible associated with the current user goal.

The first Medusa system is designed to be used by novices, since the cognitive walkthrough method was designed as a tool to explore the usability of such systems, we are justified in using it. And while we would wish to conduct empirical testing (this is discussed further in the concluding chapter), the cognitive walkthrough method has been favourably compared with empirical testing and was judged likely to provide useful data as to the usability of Medusa (Karat, Campbell and Fiegel, 1992). The cognitive walkthrough method did not compare favourably with another inspection method, *heuristic evaluation* in an evaluation of HP-VUE, a user interface to an operating system similar to Medusa, conducted by Jeffries, Miller, Wharton and Uyeda (1991). Their results, however, highlighted a known failing of heuristic evaluation (Nielsen, 1993), that the number of usability faults discovered by one evaluator will be very small. A larger number of usability faults will be found with three or more evaluators (five or six evaluators has been found to be the optimal number, more and the costs outweigh the additional usability problems found). With a single evaluator, the cognitive walkthrough method will find a similar number of faults to heuristic evaluation, perhaps even more.

May (1993) complains that the cognitive walkthrough method can be regarded as a form of guidelines evaluation, and as such will compare badly with methods that employ larger numbers of guidelines, although this was not observed in the work undertaken by Jeffries et al. (1991). Instead, we are interested in exploiting a feature of the cognitive walkthrough that May also regards as a failing, the emphasis on formal structure and the "decomposition of a task sequence to emphasise the points of the design that should be checked." (May, 1993: 11). We are concerned, as will be explored further below, with aspects of the microstructure of certain human-computer dialogue structures, the cognitive walkthrough is better suited to this analysis than other inspection methods. This concentration on the microstructure of interaction, as others have noted, risks high-level problems going unrecognised. We repeatedly acknowledge, however, that the usability of the Medusa system will not be known in considerable detail until a prototype can be developed and representative users involved in its testing. The cognitive walkthrough method, though, especially when more fully integrated with Norman's seven-stage model (it was simply used as an explanatory device above), offers a considerable advantage over other inspection methods in permitting greater analysis and discussion of the cognitive distances (Hutchins, Hollan and Norman, 1986) between intention and action, and feedback and users goals, that indicate the amount of human information processing involved in direct manipulation interaction (Rizzo, Marchigiani and Andreadis, 1997).

8.2.2 Conducting the Walkthrough Method

The cognitive walkthrough method comprises two phases of activity; a preparation phase, which is followed by the walkthrough itself. In the first phase, a set of task scenarios (which must be supported by the system under investigation) is created. For each task devised, an action sequence is created, this is a list of actions which, if

performed by users would result in the task being successfully carried out. Also for each task scenario, assumptions about the users' abilities and initial goals must be stated. For the second phase, the walkthrough itself, for each of the actions in the action sequence defined for each task scenario, a number of questionnaire forms, provided in Polson, Lewis, Rieman, and Wharton (1992) must be completed. The forms contain questions about the availability of operations and the observability and relevance of feedback in the display. In addition, the forms require the evaluator to describe how the current set of users' goals is revised as new goals are created and achieved as either the task is successfully performed, or in response to usability problems discovered. If the questions in the forms cannot be answered successfully, then a likely usability failing of the system will have been identified. Other sets of walkthrough questions have been provided, for example in (Wharton, Reiman, Lewis and Polson, 1994) and (John and Packer, 1995), but these articles seek to provide a more usable version of the method for those evaluators who are not necessarily skilled in HCI. The set of questions given by Polson, Lewis, Rieman, and Wharton (1992) obtain the most information from a system, and so this set was employed.

The cognitive walkthrough method was developed for evaluating the usability of "walk up and use" systems such as automated teller machines and information booths where the number of operations available at one time is limited, as are users' experience and prior knowledge of the tasks supported by the system. The walkthrough method has since, though, been shown to be of use in evaluating more complex graphical user interfaces, and has been shown to be capable of being learned without great difficulty by software designers who are not specifically trained in HCI (John and Packer, 1995). The greatest problem with applying the cognitive walkthrough method to systems such as Medusa, is the difficulty of only being able to step through one of possibly many action sequences that permit the task to be carried out. Where a choice of actions is accepted by the dialogue component of a system, and both actions make progress in the performance of the

task, only one of the available choices may be considered in detail. Solutions to this problem, and modifications to the cognitive walkthrough method which would permit choices of actions to be considered in detail will not be considered further here.

8.3 A Cognitive Walkthrough of the Medusa System

8.3.1 Preparation

The first phase of the cognitive walkthrough method is the preparation phase. This phase is itself composed of a number of tasks that the system evaluator must conduct. The first task for the evaluator, as mentioned above, is to choose the tasks to be analysed. These tasks must be resemble those that would be performed regularly using the final system, and must be tasks that are sufficiently supported by the system's. Normally a cognitive walkthrough does not consider tasks that must be performed using other applications in addition to the one under investigation.

The second task to be performed during the first phase of the walkthrough is to provide a task description. This description is normally at a high level of abstraction, detailing the major task to be performed and the overall change in the system's state to be brought about by performance of the task. The third task is to determine the correct sequence of actions that the user must perform for each of the tasks employed to evaluate the system. Where a number of task sequences may be judged "correct" in that the task will be said to have been performed after the last action has been performed, careful choices as to the action sequence considered are required. It can be expected that the actions performed by users will not be optimal and error-free, so a realistic action sequence must be listed.

The next task for the evaluator is to identify the intended user population of the final system. This is a task that is deliberately ignored in this case. Where the size of the intended user population of the system grows, it becomes impossible to make all but very general statements about the visual capabilities, physical impairments, education levels, cultural background and computing experience of the users. We therefore make no assumptions about potential users of the Medusa systems, apart from general population trends, and that they can be expected to have an understanding of cultures in Western industrialised societies, and can read the display contents and use a mouse without difficulty.

The final task to be completed by the system evaluator during the preparation phase is to describe the user's initial goals. That is, the system state and the state of the wider task domain that they wish to bring about by performing the task. No consideration is made of the user's wider aspirations, attitudes towards work and technology, or the basic goals they are assumed to hold in common with all autonomous systems.

8.3.2 Performing the Cognitive Walkthrough

Once the preparation phase has been completed, the walkthrough itself can be performed. The walkthrough is a repeated cycle of activity where for each action in the action sequence constructed during the preparation phase, the individual(s) conducting the walkthrough are required to complete the forms and answer the questions provided by Polson et al. (1992) for each action. These forms are reproduced in Appendix B. The questions that system evaluators must answer ask whether each "correct" action may be successfully planned and performed, whether the resulting system feedback, if any, is interpreted appropriately and useful way by the user, and consider whether progress towards completion of the task is seen to be being made.

8.3.3 Task 1 - Running an Application

The first task considered, for which a walkthrough is conducted is running an application program using the Medusa system. The Medusa system retains from other model world-based systems the categorisation of files into application programs and data files . In this task, the user wishes to run an application program without making use of any particular data file. It is assumed that the icon denoting the application is visible on-screen and has been located and recognised by the user. This first task is deliberately simplified in order to demonstrate the use of the cognitive walkthrough method. The correct action sequence for this task is given below:

1. Move pointer to application icon. [Storyboarded in Figure 8.2]
2. Press mouse button. [Figure 8.3]
3. Move pointer over **Run application** option in toolbar menu. [Figure 8.4]
4. Release mouse button.

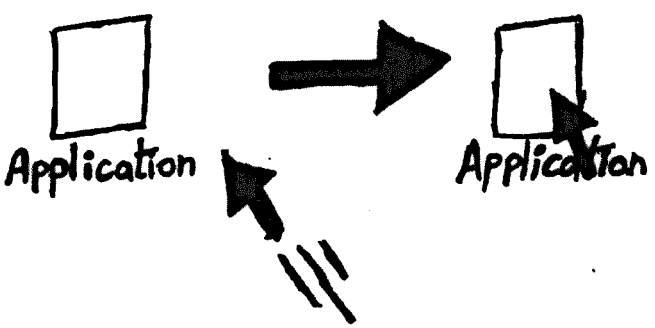


Figure 8.2 Moving the pointer over an icon

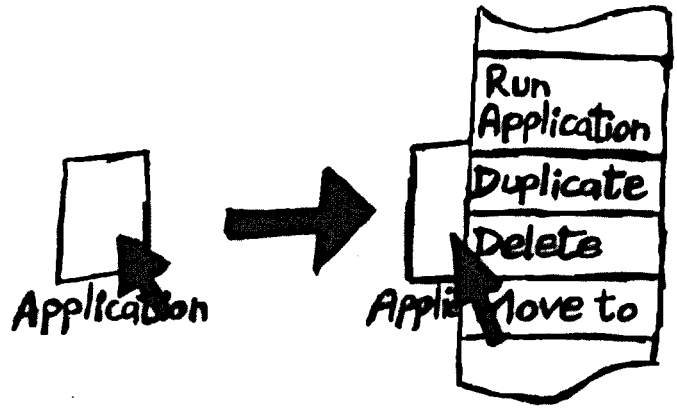


Figure 8.3 Revealing the toolbar for a file

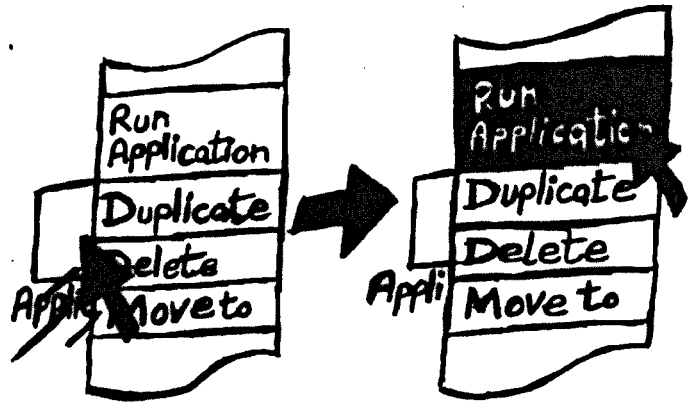


Figure 8.4 Moving the pointer over the "Run Application" toolbar option

The resulting goal structure arising from conducting the walkthrough is shown in Table 8.1 where the notation employed in Polson et al. (1992) is used. The walkthrough for the first task was completed in approximately two and half hours, written notes were taken and a verbal protocol of the author conducting the walkthrough was recorded on audio tape to allow the contemporaneous notes to be clarified and confirmed later if necessary. Protocol analysis of the recording was not required as the action sequence for the task is know and prescribed by the cognitive walkthrough method.

Run application		
	Move pointer over application icon	
and-then	Run indicated application	
		Press mouse button
	and-then	Move pointer over run option
	and-then	Release mouse button

Table 8.1 Goal structure for first walkthrough task .

8.3.4 Task 2 - Moving a File

The second task considered is the movement of files from one location to another in the Medusa file space. File spaces are abstractions over the arrangement of bytes of information stored on physical storage devices. In addition to the abstractions employed by the operating system, the location of files gives them an additional meaning in addition to their contents. The location of a file in the file space can denote its meaning in the user's working history, their past projects, their on-going work, and the resources they are employing in their immediate tasks. The task for which a walkthrough is conducted is the movement of a single file from one directory to another. The correct action sequence for this task is listed below:

1. Move pointer over icon denoting file to be moved.
2. Click mouse button.
3. Move pointer over **Move to...** option on toolbar. [Figure 8.5]
4. Click mouse button.
5. Move pointer to icon denoting the intended destination container or window view onto a directory listing file. [Figure 8.6]
6. Click mouse button.

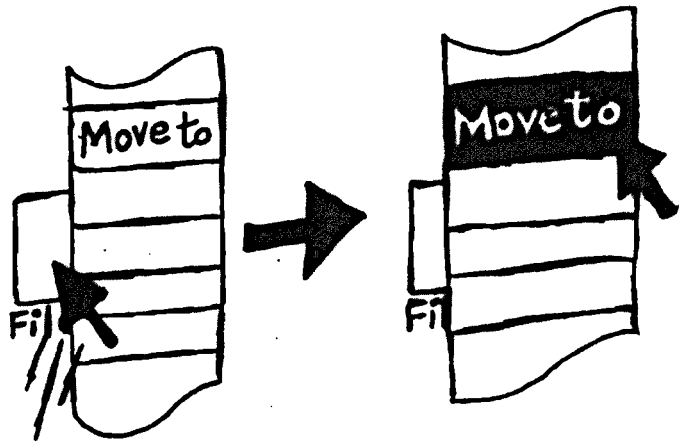


Figure 8.5 Selecting the "Move to" toolbar option

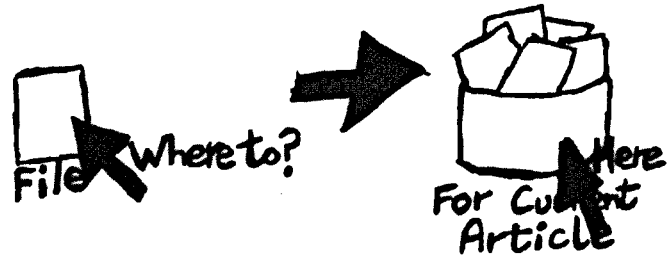


Figure 8.6 Indicating the destination container when moving a file

For this task and the next one considered, we alter the basic level of interaction to use the cognitive walkthrough to determine whether the structure of basic selection tasks that make up interaction with the Medusa system should be performed by the "press mouse button → move pointer over option → release mouse button" action sequence found in the Apple Macintosh desktop and environments for programming in Smalltalk-80, or by the "click mouse button → move pointer over option → click mouse button" sequence found in Microsoft Windows and some X Windows toolkits. The goal structure eventually constructed for this task, defined using the notation of Polson et al. (1992) is shown below in Table 8.2.

Move file of interest to intended location		
	make toolbar visible	
		move pointer to icon
	and-then	click mouse button
and-then	select option from toolbar	
		move pointer to **Move to..** option
	and-then	click mouse button
and-then	specify destination	
		move pointer to destination
	and-then	click mouse button

Table 8.2 Goal structure for second walkthrough task.

The walkthrough for this task was completed in two hours, hand-written notes were taken during the walkthrough and a verbal protocol was recorded onto audio tape.

8.3.5 Task 3 - Adding a Method to the Toolbar

As described in Chapters 6 and 7, commands are directed towards on-screen objects via the toolbar. The options available on the toolbar at any point in time are determined by the category of object towards which the command is issued, the object's current state, and the options selected by the user to be present on the toolbar in certain situations. The Medusa system, through the on-screen display described as being part of the computer-computer metaphor, allows the user to modify the behaviour of the toolbar. This should be contrasted with other menu-based systems where menu options can be used during interaction, but the user has no meta-reference to menu options themselves, and is unable to modify them. The third task we consider is the addition of a previously unavailable option to the toolbar. This

option will only appear on the toolbar when appropriate when interaction with the toolbar, as opposed to the "meta-toolbar" is resumed. The action sequence for this task is given below:

1. Move the pointer over the "meta-toolbar" in the on-screen computer-computer display. [Figure 8.7]
2. Press mouse button.
3. Move pointer to **Add command** option in the toolbar. [Figure 8.8]
4. Click mouse button.
5. Move pointer over **Edit using Text Tool** option. [Figure 8.9]
6. Click mouse button.

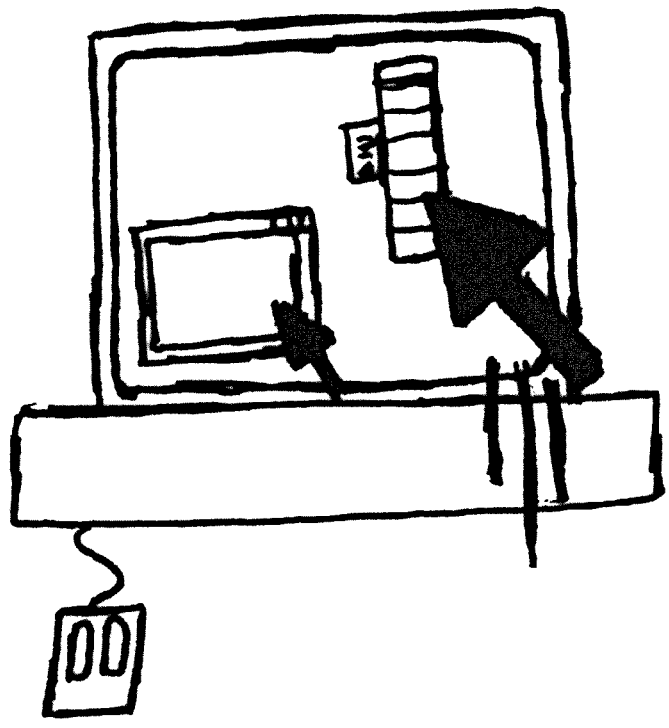


Figure 8.7 Moving the pointer over the meta-toolbar

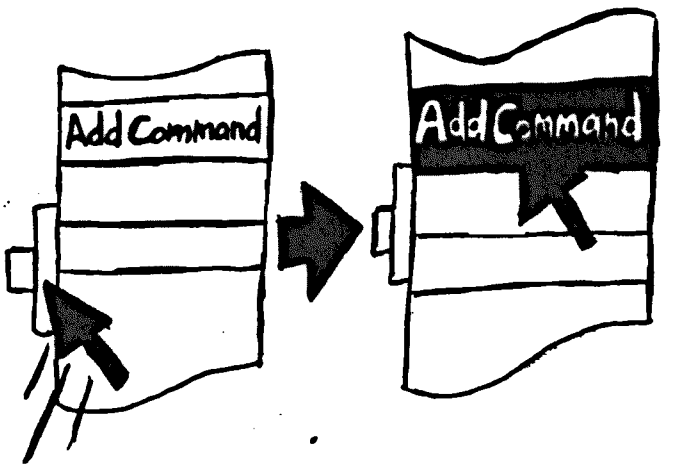


Figure 8.8 Selecting the "Add Command" toolbar option

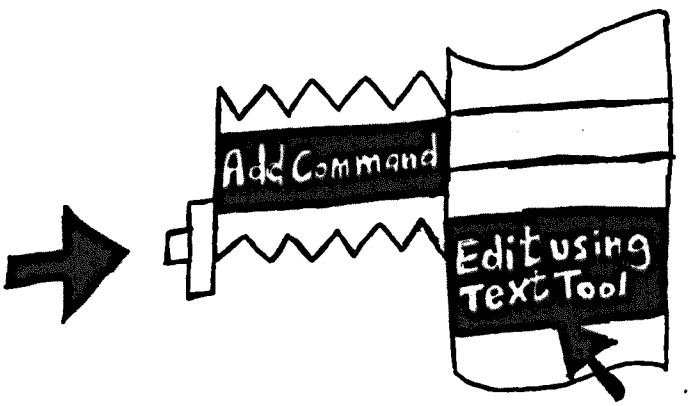


Figure 8.9 Selecting the "Edit using Text Tool" hierarchical toolbar option

As with previous tasks, a goal structure for the task was constructed, experience gathered from conducting the second walkthrough allowed a structure that made greater use of hierarchy based around the smaller selection tasks that make up the larger task to be constructed. The goal structure constructed in the Polson et al. (1992) notation is shown in Table 8.3.

Add option to toolbar			
	make toolbar visible (direct messages to meta-toolbar)		
			move pointer to meta-toolbar
		and-then	click mouse button
and-then	add option to toolbar		
			move pointer to **Add**
Command..			
		and-then	click mouse button
		and-then	move pointer to **Edit using**
Text Tool option			
		and-then	click mouse button

Table 8.3 Goal structure for third walkthrough task.

8.4 Design Flaws in the Medusa System Version One

While intended to overcome the usability difficulties of existing metaphor-based systems, the cognitive walkthroughs conducted on the first design of the Medusa system reveal that this system too suffers from some usability failings. Some of these failings reveal an interesting shortcoming of the principle of visibility underlying the system design, and reveal that even the notion we have described as letting the computer act as a metaphor for the target computing system may be subject to breakdown, and a failure in its scope.

8.4.1 Basic Interaction

Like other WIMP systems, the Medusa system assumes that the large part of interaction with the system will be accomplished by using an input device capable of generating location and selection information. A location, in the form of two-

dimensional co-ordinates , can be generated by an input device such as a mouse or trackball, in the case of the Medusa version one system, the choice of devices is of no concern, only the *design space* (Macinlay, Card, and Robertson, 1990) is relevant. Selections are made using a single switch or button; in Medusa version one, most tasks are selection tasks, and no more complex form of input is required.

Interaction with many objects in the Medusa model world is based on an object-message notion, on-screen objects change their state in response to messages (often in the form of options on the toolbar) being sent to the object of interest. As explained in Chapter 7, the metaphor of highlighting is not adopted in the Medusa system due to the problems that it causes users. Instead clicking on an object is interpreted as a cue to present the toolbar listing the commands or messages that may be issued to that object in its current state. This mechanism permits the distinction between *tracking* and *naming* objects to be exploited within Medusa, the recognition that one need not identify an object in order to be aware of its presence or type in order to perform some tasks upon it (Smith, 1996). Medusa allows this distinction to be employed within the model world as it can in the real world. In order to rely on the association in the user's mind that the toolbar that appears refers to the object that the pointer was over when the mouse button was clicked, two explanations can be given. The first is a transfer of the notion of the *current object* (similar to the notion of *focus*, the client application to which subsequent events should be directed) by those users that understand the concept from their experience of using other window-based systems. The second exploits the phenomenon, claimed by Jeannerod (1997) to have been first documented by Aristotle, that events arising from the same region of space in the visual field apply to the same object.

This approach to interaction differs from interaction with the options listed on the toolbar and its associated menus. On the toolbar, an option becomes highlighted when the pointer's hotspot is over it. Two reasons are proposed for this difference.

One reason is pragmatic, deriving from a need to be aware of the applicability of Fitts' Law in HCI, the target of a toolbar option is small, as is the hotspot (the active region of the pointer, the location of which is taken to be the source of a mouse event), highlighting an option allows the user to confirm that the intended target of a movement of the pointer has actually been reached. The second reason is that we claim that the user selects toolbar options whereas objects in the model world are not selected, there is no notion of a current object, only one which is the object to which a message is sent. The first Medusa system thus employs an *object-message* interaction style, rather than choosing between SVO, SOV or VSO structures (Subject-Verb-Object, Subject-Object-Verb, and Verb-Subject-Object respectively) that categorise languages (Pinker, 1994), including direct manipulation user interface dialogue structures, thus removing the need to be aware of the culture into which the system is to be introduced. By avoiding the conversation paradigm, it is possible to consider further the continuum of model worlds from two dimensional model worlds to augmented and virtual realities, this will be explored further in Chapter 9. Users must, however, learn the basic action sequences that make up selection tasks in Medusa. These action sequences may differ from the low-level motor sequences that users have mastered when using other systems, although they are also easily described using the dialogue modelling approach of Buxton (1990).

One question, as mentioned above, that it was hoped that the walkthroughs would answer was which dialogue structure making up selection tasks in Medusa should be chosen. It would appear that neither choice makes much difference to the usability of these very basic tasks. The only problems that require further designer effort are those that arise when users familiar with one approach transfer the motor sequence to a system that uses the alternative approach. An approach to overcome this problem, one adopted by the Solaris desktop environment, is to support both alternatives within the state machine for processing mouse-generated events wherever possible. Different routes may therefore be taken to reach the same final (accepting) state of

the dialogue's state machine that is interpreted as a toolbar option selection task having been performed.

8.4.2 Understanding the Computer-Computer Metaphor

The cognitive walkthroughs revealed few usability problems that arise from interaction with the on-screen components whose design is motivated by the notion of the computer-computer metaphor. The problems that were revealed are ones of inelegance motivated by the requirements of consistency and visibility. The addition of methods to, or deletion of methods from, the toolbar in the third walkthrough requires that the user select methods from a complete list of all the methods to objects of all categories that may be encountered within the system. This arises because the "meta-object" in the computer-computer on-screen display is of no particular category (in the Medusa ontology it is simply an "object"), thus the list of methods from which selections can be made cannot be restricted to those applicable to a particular object category. While it might be possible to have the user select an object's category and then modify the toolbar, such a solution would require the task to be performed within a dialogue box. In this solution the object's category could be selected, and a choice from the subsequent list of associated methods made. The use of dialogue boxes, however, requires that sub-dialogues that cannot be interrupted be implemented. The user will not be able to complete other tasks or respond to urgent events or alarms until the task supported by the dialogue box has been satisfactorily completed. It is possible to refer to environments such as ARKola to show the advantages of the approach adopted there and also here in Medusa.

8.4.3 Directly Manipulating the Intangible

The first version of the Medusa system seeks to provide user interface components that represent important aspects of the underlying functionality sufficient for a more

complete understanding of the system to be obtained than can be obtained from other metaphors in the depiction of the model world. The results of the cognitive walkthroughs, however, suggest points where even the scope of letting the computer act as a metaphor for the underlying computer system is limited, and where tasks are not easily supported by the Medusa user interface design. These points of breakdown arise from the need to provide on-screen representations of data structures that are not provided as part of the underlying operating system, but which are part of the interface itself. These representations depict data structures that do not describe a system state, but instead they describe the user's interaction with the system, often referring to past events, not the current state and states which might be achieved in the future.

One problem arises from the basic means of interaction with the system. A task supported by many existing window-based systems is the use of a double click of the mouse button. The double click usually performs the "open" task, to which the system responds by displaying the contents of a folder, running an application or restoring an icon depicting a running application to its full-sized window representation. The use of the double-click is ubiquitous, but problematic. A double-click is an event invested with greater semantics by the system than the combination of two mouse button clicks separated by a short delay that makes it up. Olsen (1998) suggests that this difficulty of interpretation of events is resolved by the first `button-down` event being interpreted as selection of the object, and the second click as opening the object. The double click thus presents the Medusa system with a number of problems. Firstly, it presents the problem of depicting the notion of the currently selected object, which we seek to avoid entirely. Secondly it presents the problem of supporting the notion of "opening" a file which relies on a metaphor within the model world mapping to a very wide range of different system semantics. Due to the range of speeds with which users can, and prefer to, double-click the mouse button, interface features are often provided allowing the user to place an

upper limit on the delay between clicks below which the sequence of actions is to be interpreted as a double-click event. While representations allowing users to adjust this value can be devised, the delay between click events is a value which must be stored in a data structure that is introduced by the development of a user interface. The designer is not required to provide a user interface to an existing data structure, or to support the user's existing work practice, instead the must introduce new concepts that are not derived from either of these domains.

Direct manipulation user interfaces, as has been noted above, are characterised by easily reversible actions. The lessons drawn by Polson and Lewis (1990) from their CE+ model of interface learning include the need for systems to provide obvious ways for the effect of actions to be undone, if the system is to be easy to learn. Many other authors have made similar requirements of interactive systems. The first version of the Medusa system does not consider how undo facilities should be provided. The provision of an undo facility within the Medusa system causes greater problems than coping with the double-click as discussed above.

Undo facilities cause particular problems for a metaphor-based system. Whereas a typical user interface metaphor presents icons in the model world that denote or depict aspects of the underlying data structures, or the functions that apply to these data structures, an undo facility is not usually a feature provided by the underlying operating system. The problem of providing an undo facility in a metaphor-based system is one of providing the facility *in the first place* in addition to providing a depiction and a behaviour of the depiction in the model world. The product-oriented view of metaphor is not one that can be adopted therefore. Providing an undo facility is complex because no clear data structure to which a metaphor is required exists, instead an undo facility must interact with a structure that represents and captures aspects of the user's *dynamic pattern of interaction* with the system. This structure is required to store the system state, methods for undoing those user operations that are

undoable, as well as the user's task and command history. This structure therefore does not fit in either with the process-oriented view of metaphor. The undo facility, while it may be guided by the user's current work practice, cannot be entirely specified and depicted from an analysis of the way in which the user's tasks are currently performed and from the language with which users describe their artefacts of work. We consider an undo facility for the Medusa system, and how undo facilities in metaphor-based system may be developed in general, in the final chapter.

8.5 Conclusions

In this chapter a critique of the first Medusa system design was undertaken using the cognitive walkthrough method. Of concern to evaluators when employing any usability evaluation method are the number of usability errors revealed by application of the method, and the assumptions about systems, users, and interaction upon which the evaluation method is based. These assumptions determine the *types* of usability errors that can be revealed. The numbers and types of usability errors that can be revealed by the cognitive walkthrough method have been previously examined (Bell et al., 1991; Jeffries et al., 1991). This work demonstrates that the walkthroughs conducted examining the usability of the Medusa system reported on above are likely to only reveal a small majority of the system's usability errors and that some major usability problems may be missed. One means of increasing the number of usability errors detected, by making a number of those conducting the walkthroughs experts in the task domain supported by the software application is clearly not possible for the sort of system considered here where no particular real world task domain is supported.

The cognitive walkthroughs conducted reveal no other usability problems, but we can predict aspects of the system that might give rise to difficulties in a full

implementation. The design principle of making relevant data structures visible will sometimes be in conflict with the concept of tangibility. Some on-screen objects will be sources of information only, they will not be *equal opportunity* (Runciman and Thimbleby, 1986) in allowing their state to be changed *both* by the system and the user. An example is the keyboard buffer which shows the text that has been typed by the user but which has not yet been processed by the target application. The effect of further typing (even of presses of the delete key) will serve only to add to the contents of the buffer, the semantics of keys such as the delete key must be provided by the application. It therefore makes no sense for cut, copy, and paste tasks to apply to this display, even though it resembles user interface components that might support such tasks. The consistency of interaction sought will still hold though, being an on-screen object the buffer can still be sent messages, the set of messages will be smaller, however, than the user might expect.

8.5.1 Is The Computer Metaphor Better Than Others?

One question that needs to be answered is whether the computer-computer metaphor is an improvement over existing user interface metaphors. The methods employed here show that some tasks are no more difficult to perform in Medusa than in other systems, but that other tasks have more complex action sequences in order to be consistent with the design approach adopted. The consistent approach adopted however means that once a correct interaction sequence has been learned, it can be employed when interacting with all categories of on-screen objects. In the first Medusa system, our concern is that the semantics of operations can be easily learned by the user. The theoretical framework used to examine the system semantics is silent on how operations are invoked or performed, hence the system usability — in terms of putting intention into action — may be found lacking, although we found no serious difficulties from the testing conducted. In the other versions of the Medusa system described in Chapter 9, the nature of the actions needed to perform in order

to change the state of the system has received considerably more attention. Whether this attention results in improved usability is not yet determined.

Chapter 9

Revised Versions of the Medusa System

"As long as the software is nerdified, and major conceptual limitations are built right into the software at that level, then it cannot get far. This is a philosophical question: when people program - i.e. decide on which set of possible options they should make available - they express a philosophy about what operations are important in the world. If the philosophy they express is on anything like the level of breathtaking stupidity that the games they play and the internet conversations they have are, then we are completely sunk."

— Brian Eno (1996) *A Year with Swollen Appendices*, Faber and Faber.

In previous chapters, serious criticisms of the world view underlying the theories of metaphor assumed to be employed in user interface design and human-computer interaction were presented. This world view can only be assumed, however, as few design case studies or articles on metaphor in the human-computer interaction literature of which we are aware explicitly state the theory of metaphorical comprehension, extension, or mapping, employed in the design of a particular model world. Previous chapters also served to introduce a recent theory of metaphor (the Lakoff/Johnson "contemporary" theory) and to explicitly apply it to user interface designs. The case studies examined features of existing user interface designs that any theory must be able to explain given that these systems are used successfully by users, or which give rise to documented difficulties attributed to users' failures to recognise, comprehend, and make use of the metaphor. The Lakoff/Johnson theory

was found to satisfy these requirements of a theory of metaphor when applied to graphical user interfaces.

9.1 The Medusa System - Version Two

The first version of the Medusa system, described above, is one which uses the methods of analysis employed in Chapters 4 and 5 as the basis for describing objects in the model world and designing tasks that change the state of these objects. The world view on which these means of analysis are based, however, is one that is rejected in Chapter 4. In addition to employing the Lakoff/Johnson theory of metaphor as a tool for *analysing* existing user interface designs, if it is to be judged worthy of further consideration, it should be employed as a means of *generating* user interface metaphors that can be more readily comprehended by the user. The design of a second, revised, version of the Medusa system is thus presented in this section.

9.1.1 Direct Manipulation

It is clear that in order for users to be able to manipulate on-screen objects, and perform operations on them, users must be able to recognise the on-screen arrays of pixels as distinct objects. The objects having attributes and functionality provided by the underlying software. Users must also be able to classify them so they suggest what actions may be performed on and using them. The action sequences that can be performed will come either from metaphorical extension from other graphical user interfaces, prior experience, or from some form of instruction or help in using the system. It is this aspect of the design of metaphorical model worlds that led to consideration of a contemporary theory of user interface metaphors after traditional views of categorisation were found subject to the same assumptions and problems of an Objectivist world view. The modern view of categorisation is not adopted in the

Medusa version one model world, although subsequent versions of the Medusa system take this into account.

As well as having to categorise on-screen objects to make use of them, it is necessary to be able to categorise events and actions in a model world. The most important category in learning and using an interactive system is that of causality. This is the perception of a user-initiated action causing feedback or a change in attributes of the object that the user directly interacts with, and in any objects and attributes that the user additionally interacts with indirectly during the course of their action. Lakoff (1987: 54-55) observes that:

> "Prototypical causation appears to be direct manipulation, which is characterized most typically by the following cluster of interaction properties:
>
> 1. There is an agent that does something.
> 2. There is a patient that undergoes a change to a new state.
> 3. Properties 1 and 2 constitute a single event; they overlap in time and space; the agent comes in contact with the patient.
> 4. Part of what the agent does (either the motion or the exercise of will) precedes the change in the patient.
> 5. The agent is the energy source; the patient is the energy goal; there is a transfer of energy from agent to patient.
> 6. There is a single definite agent and a single definite patient.
> 7. The agent is human.
> 8. a. The agent wills his action.
>
> b. The agent is in control of his action.
>
> c. The agent bears primary responsibility for both his action and the change.

9. The agent uses his hands, body, or some instrument.
10. The agent is looking at the patient, the change in the patient is perceptible, and the agent perceives the change."

In previous chapters aspects of behaviour of existing user interface designs that cannot be categorised as direct manipulation, and that could not be accounted for by a metaphorical mapping to the physical world were discussed. A criteria of the Medusa interface design is that such user interface behaviours should be avoided, and that it should be possible to categorise user actions as direct manipulation according to Lakoff's definition. The first version of the Medusa system above simplifies interaction mostly to selection tasks. While this design choice attempts to ensure consistency and to prevent breakdowns in the system image, methods of interaction and action sequences familiar to users from other flat model worlds cannot be applied fully, but designs of Medusa interfaces cannot ignore transfer between systems.

9.1.2 The Workbench

The use of the root window in the second version of the Medusa system does not differ from its use in the first version, so little will be added in this section to that given in Section 7.2.1. The root window in this version of Medusa remains a *workbench*, an area for planning that allows the user to place objects and groups of objects in positions of their choice before using the objects in their tasks, or placing them in the intended final destination. Kirsh (1996) describes some ways in which the environment may be used in planning and performing tasks. An important use of space is to permit better choices of actions to be made, and to serve as a source of reminders, while tasks are being performed. By constraining the perceived *action set*, the actions seen as being possible at a particular moment in time, affordances may be simultaneously constrained and highlighted. The ways in which on-screen

objects can be positioned so as to aid planning and task performance are sometimes restricted by the user interface style. The use of windows and the restriction, by some systems, of icons to a fixed grid of locations physically limits the placement of icons on the 2D desktop. Kirsh (1996: 419) also suggests that simple linear arrangements of objects to be employed in a task sequence are too restrictive. He claims that even production line assembly plants do not employ strictly linear arrangements of objects to be manipulated, and that agents must usually rely on "known systems of arrangements, or on some design that makes sense relative to the subject matter." Kirsch suggests that the DESKTOP metaphor encourages the placement of peripheral equipment such as printers and the wastebasket around the edges of the screen to reflect the traditional placement of office equipment around office walls. Such placement might not be best suited to the needs of the user in performing tasks involving on-screen objects. The location of objects also impacts upon the tasks of seeking wanted icons and determining their location. Kirsh (1996: 422) says:

> "Perhaps the most obvious way of simplifying perception is to arrange objects in space so they form equivalence classes, or partitions, that reflect preconditions, or properties that are useful to track, notice or exploit...The primary value of such external partitioning is that it makes it easier:
>
> - to keep track of where things are;
> - to notice their relevant affordances."

9.1.3 Objects in the Model World

On-screen objects in the model world of the first Medusa system are instances of categories of objects defined in a hierarchy following the notions of classical categories, that is, membership of a category (or class, in the implementation) is

determined by the object possessing necessary and sufficient attributes. The Medusa object category structure shown in Figure 7.1 is artificial and imposed by the system design, although a similar approach to describing the world may be found in the ontology of Douglas Lenat's CYC system, as described by Sowa (1995). Barsalou (1995: 168) states that:

> "... clearly, the purpose of categorization is not to know an entity's category. Instead, the purpose of categorization is to identify information in memory that provides useful inferences. Upon accessing a category for an entity, a tremendous amount of knowledge becomes available that is useful in a variety of ways. This knowledge may specify the origins of the entity, its physical structure, its possible behaviour, its implications for the perceiver's goals, or actions for interacting with it successfully. Accessing a category is not an end in itself but instead the gateway to knowledge for understanding an entity, and interacting with it properly."

The first version of Medusa imposes a classical category structure on objects in its model world, as do most, if not all, object-based interfaces. The classical theory of categories has been questioned, however, for at least five decades, and the first Medusa system ignores the fact that category structures can be created to meet the needs of performing immediate tasks and to achieve short-term goals (Barsalou, 1995). Many of these novel categories are based around prototypes (clear cases and best examples of category membership), in accordance with the modern theory of categorisation (Rosch, 1973, 1978) employed and described by Lakoff (1987). It remains a matter of further study as to what role category structures play in understanding and interacting with graphical model worlds. The example of GIF format files was discussed above as problematic in that most are still images, but some, as web-page programmers know and take advantage of, make up small

animations. It would be interesting to know if a radial category structure, of the sort employed by Lakoff (1987), based around prototypical still and animated files forms any part of users', or perhaps only web programmers', understanding of file systems. The second version of Medusa seeks to acknowledge the modern theory of categorisation. Previous attempts to accommodate prototypes in this theory with object-based and object-oriented user interface design have been unsatisfactory, however, as object-oriented design adopts the classical theory in the categories that make up class hierarchies. A better approach to the implementation of the second Medusa system is to make use of programming languages with a PROTOTYPE-INSTANCE object structure rather than the traditional CLASS-INSTANCE approach. In this way the user can more easily impose a category structure on objects in the model world than they can in the first version of the Medusa system.

9.1.4 File Management — Piles of Objects

The file management facilities provided by the first version of the Medusa system were developed to resolve failings in implementations of the desktop metaphor. In particular they address the DESKTOP metaphor's use of files and folder analogies as a means of accounting for the structure of the file space and the tasks that alter the state of the file space. The folder is only one form of possible file organisation, however. Despite argument to the contrary (Fertig, Freeman, and Gelertner, 1996), the conclusion reached by Nardi and Barreau (Barreau and Nardi, 1995; Nardi and Barreau, 1997) is that users prefer location-based search of files, and that locations of files serve as reminders of tasks to be performed. They also state that most users archive relatively little information and avoid elaborate filing schemes. Their proposed requirements for filing systems and filing tasks are thus not satisfied, and indeed are made more difficult, by the traditional notion of the desktop metaphor.

Malone's (1983) study of how documents and information resources are arranged in the physical office differentiates between files and piles. In Malone's terminology *files* are defined as units where the elements (such as individual folders) are explicitly titled and arranged in a systematic order (such as alphabetical or chronological). Groups, such as drawers in filing cabinets, may also be explicitly titled and systematically arranged, but they need not be. In *piles*, though, the individual elements (papers, folders, and so on) are not necessarily titled, and they are not generally ordered in a particular way. Table 9.1 summarises differences between files and piles.

	Elements titled	Elements ordered	Groups titled	Groups ordered
Files	Yes	Yes	?	?
Piles	?	No	No	?

Table 9.1 Units of desk organization (Malone, 1983: 106)

File management in the second version of the Medusa system is intended to satisfy the following criteria, or to take account of the following observations:

1. Categories are often devised to suit the needs of tasks and such categories are often based around prototypes.
2. Objects can be placed and grouped in locations so as to distinguish them from other objects, to highlight their affordances, and to aid in the performance of users' tasks.
3. The folder metaphor restricts the placement of objects and is prone to breakdowns.

4. The Lakoff/Johnson theory of metaphor comprehension claims that meaning is structured and grounded in image schema that capture our repeated and common experiences of interacting with the external world.

The second version of the Medusa system therefore adopts a means of organising files based on piles, and tasks to interact with piles are grounded in image schema to obtain their meaning. A file organisation system based on a pile metaphor has already been devised (Mander, Salomon, and Wong, 1992). In this implementation of a pile metaphor, the folder metaphor is not adopted, instead files are arranged in pile structures, these piles can be casually organised on the root window. Electronic piles can be either system-created or user-created. System-created piles are stacked neatly, implying a set of rules behind how the pile was created. User-created piles have a dishevelled appearance, items are added to the pile by being "dropped" onto it. Piles may be labelled (to indicate categories, possibly relevant to the user's current tasks). When a file is dropped onto a pile a dialogue box is presented asking whether the file should be simply added to the pile, or whether the user wishes to modify the script that was employed when constructing the pile. Over time the criteria by which it is appropriate for a file to be placed on the pile may change, hence the need to change the script may change. At its simplest, a file may be placed on the pile because the file contains particular keywords. Other, more complex, placement strategies require that the user is able to write scripting language programs that determine how piles are constructed.

The pile metaphor devised by Mander and his colleagues adopts the product-oriented approach to metaphor-based design. The piles do not depict existing data structures within the operating system's functionality (as in the case of the treatment of files, links and directories in the first version of the Medusa system), instead the piles depict data structures added to the desktop's functionality. These data structures are not necessarily based on the user's work language and task processes, they are only

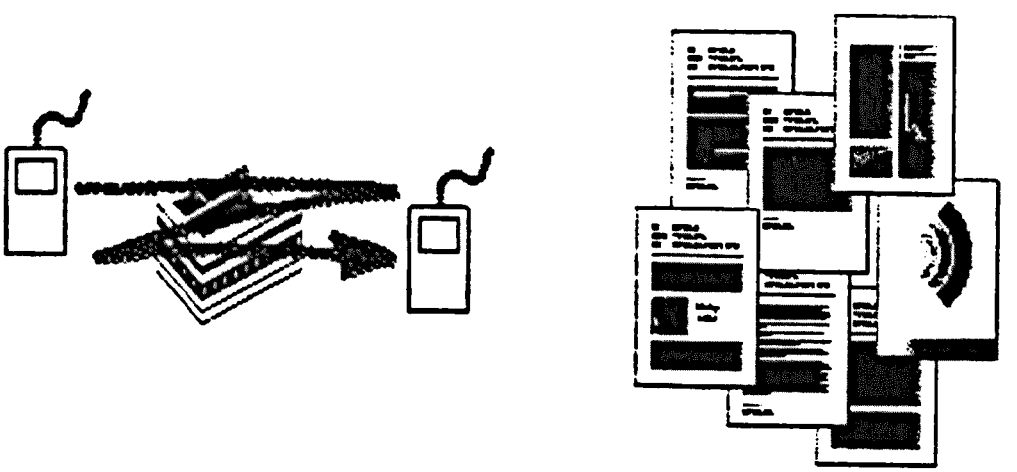
based on the broad results of the need to support casual structuring of files noted by Malone (1983). The resulting design process adopted by Mander and his colleagues was to construct runnable prototypes of paper-based designs using Macromedia Director which were then examined by users.

The testing undertaken of the prototype piles addressed the piling models, the methods for initiating browsing, viewing cone representations and how items are found within a pile. An issue that will be addressed further below is the piling model, whether piles are "document-centred" or "pile-centred". In the document-centred approach, the pile is represented as a collection of individual items, each document is depicted by a rectangle, a pile being created whenever a single document on the desktop has another placed on top of it. Items on the pile may be removed by clicking on any visible region of the item of interest and dragging it away from the pile. The pile as a whole, however, cannot be moved to a new location on the desktop. In the pile-centred approach, a pile acts like a Macintosh folder. If a dragged file passes over a file on the desktop, the occluded file is highlighted (like a folder) to indicate that it is a potential target and that a pile would be formed if the user were then to drop the held file onto the file below. Clicking on any part of the pile and dragging the pointer moves the entire pile around the desktop.

In user testing, Mander and his colleagues (1992) found that while individual users displayed a preference for one or other of the pile creation methods, neither was judged to be superior. A number of problems were, however, revealed. In the pile-centred approach, users appreciated being able to add objects to the pile easily, and being able to move the pile as a whole, but noted the problem of selecting an individual item from within the pile. The opposite is true of the document-centred approach where users were unclear as to how to move the pile as a whole, but users appreciated being able to easily select an individual item from within the pile. In

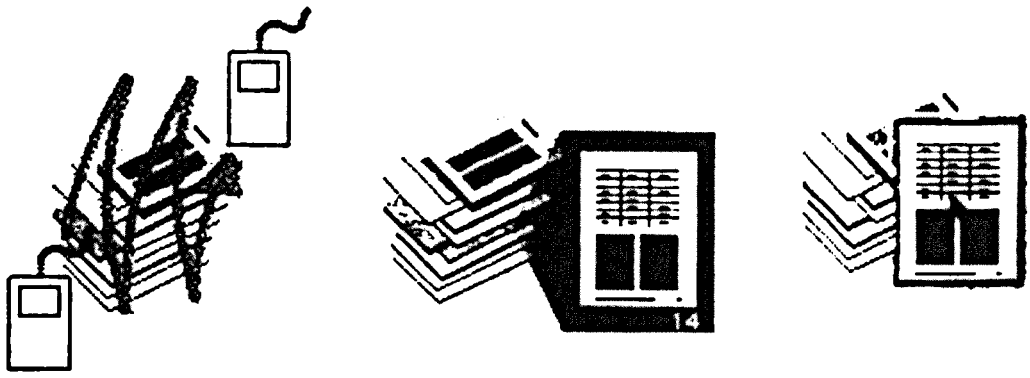
both cases users had the difficulty of knowing whether a file dropped onto a folder sitting on the desktop would be placed inside the folder, or whether a pile would be created. Experience of the Macintosh interface was found to lead most users to believe that the file would be placed inside the folder. As Mander, Salomon, and Wong (1992) note, the design of piles and user's expectations cause questions to be raised about how well the pile metaphor fits into the desktop metaphor.

Another important issue, which will be addressed further below, is the issue of interaction with piles. This concerns how Mander and his colleagues solve the difficulties of emulating interaction with complex fragile 3D structures in the real physical world, in an environment where interaction is limited to gestures that can be generated using a mouse or touch screen. Testing compared two approaches, between double clicking and a horizontal gesture (shown in Figure 9.1) to spread out the pile's contents, and between double-clicking and a vertical gesture (shown in Figure 9.2) to browse the pile's contents within the viewing cone.



(a) Mouse gesture (b) Spread-out contents of pile

Figure 9.1 Spreading out a pile's contents by a horizontal gesture (Mander, Salomon, and Wong, 1992: 630).



(a) Gesture to generate viewing cone

(b) Viewing cone showing a page of the document under the pointer

(c) Document selected and removed from the pile

Figure 9.2 Gestures to browse the contents of piles

The results of the testing undertaken found that 9 out of 10 users preferred double clicking on piles over the use of gestures, subjects finding the gestures non-intuitive and ambiguous. In general, users stated that they would employ the "spreading out" approach to viewing the contents of piles, this approach better supporting comparison and recognition tasks.

Failings of the Pile Metaphor

To understand the failings and successes of the pile metaphor requires completion of a larger exercise, an exercise one can describe as an effort to understand the fabric of meta-reality[1]. Meta-reality, a term coined by Smith (1986) and depicted in Figure 9.3, is the space in ARK in which the hand resides and in which objects removed from ARK alternate realities reside until replaced into a possibly different alternate reality. A similar space can be found in each of the interfaces described in Chapter 2. It is the space that the pointer (or hand) resides and moves within. The pointer is an

[1] So termed by analogy with David Deutsch's (1997) *The Fabric of Reality*, Penguin, London.

on-screen object that only needs to exhibit spatiomimesis, and that can be subject to breakdowns in spatiomimesis due to temporal uncertainties in the underlying hardware and software. The *reality* (or desktop, or room network, and so on) beneath the meta-reality must display a larger repertoire of behaviour, and is subject to a wider range of breakdowns. The pile metaphor is another metaphor subject to breakdowns, for example, objects dropped onto electronic piles do not bounce off and fall to the table, and electronic piles are stable no matter how high they are built and do not topple over. The pile metaphor, perhaps more than other 2.5D model worlds, reveals that an account of understanding of such user interfaces must account for a reality/meta-reality split. We observe that the design space of the mouse used to position the pointer is a 2D plane, but the reality beneath is 2.5 or 3D, for example, no matter how "high" electronic piles grow, the pointer never collides with them.

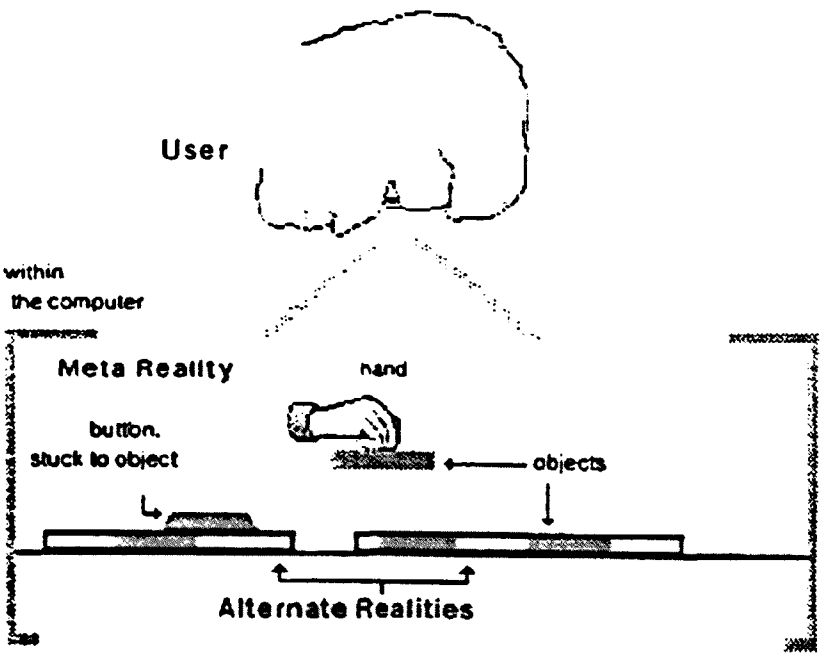


Figure 9.3 Reality, alternate reality, and meta-reality (Smith, 1986).

Both folders (called files by Malone) and piles pose additional problems that are not part of the metaphors that are used to understand them, but which must be solved to allow tasks to be performed on them. The principle concern in this thesis is to

provide facilities that allow objects in the on-screen model world to be arranged to aid users in the performance of their immediate tasks. The user should be able to arrange information resources (files in traditional computing terms) to aid reminding and to constrain and suggest actions. Malone's study reveals that piles, in themselves, do not aid with identifying the priority of tasks. Malone suggests that the colour of items on the electronic desktop could denote their priority, a system feature that can be found in recent versions of the Macintosh Finder running with colour display hardware. Malone suggests an alternative means of denoting priority is the size of icons, this suggestion is hard to integrate into a system such as Medusa where the physics of the model world is intended to be plausible and suggest grounding in image schemata familiar from interaction with the real world. A mapping of LARGER ↔ HIGHER PRIORITY has less meaning than mappings such as HIGHER PITCH ↔ MORE (the familiar UP is MORE schema) mentioned above. The other alternatives that Malone suggests to indicate priority are the location of items, a criteria that the second version of Medusa is designed to support, and frequency-based reminding, a task best delegated to an assistant.

9.1.5 Performing Tasks in Medusa Version Two

The first version of Medusa, described above, adopts the convention of the folder metaphor for file organisation, which has the advantage of allowing hierarchical categorisation and organisation, but the failings of the folder convention and difficulties in understanding the concept of files were addressed and hopefully resolved. The folder convention was found lacking in the need to support ad hoc categorisation of on-screen objects to perform tasks, and in the association of category with physical location that the second version of Medusa was designed to support. An obvious user interface design that supports these facilities (one assumed in the second version of Medusa) is the *pile*. This is the second approach to file organisation in information-rich work identified by Malone (1983) and which has

already been prototyped and subjected to some user testing (Mander, Salomon, and Wong, 1992).

Unfortunately, as is revealed by the analysis of interaction with piles presented in Appendix C, the image schemata that ground users' understanding of folders (in the terms of the Lakoff/Johnson theory) are the same that ground understanding of interaction with piles. A consequence, of recognising that understanding and interaction with both piles and folders are grounded in the same schemata via metaphors with similar mappings (FOLDER is CONTAINER, and PILE is CONTAINER) are that interaction with piles in Medusa version two is similar to interaction with folders in Medusa version one. This is especially the case as actions can only be expressed via the narrow channel of the mouse in our current designs.

Piles in Medusa Version Two

The design of the second version of Medusa adopts the pile metaphor as its file organisation mechanism in the pile-centred form. The folder metaphor is not adopted in this system design. The folder metaphor is part of the wider OFFICE metaphor and has the difficulties described above in supporting multimedia file types. Adopting the pile metaphor also resolves the problem of ambiguity as to whether a pile, or a folder on top of a pile, is the target for a file being moved, without the need for more complex mouse gestures or multimodal input. In a pile-oriented version of Medusa, a pile could be created explicitly by informing a file that is not currently a member of another pile or too close to other files on the workbench that it is the first element of a new pile. A toolbar option that would implement this is shown in Figure 9.4.

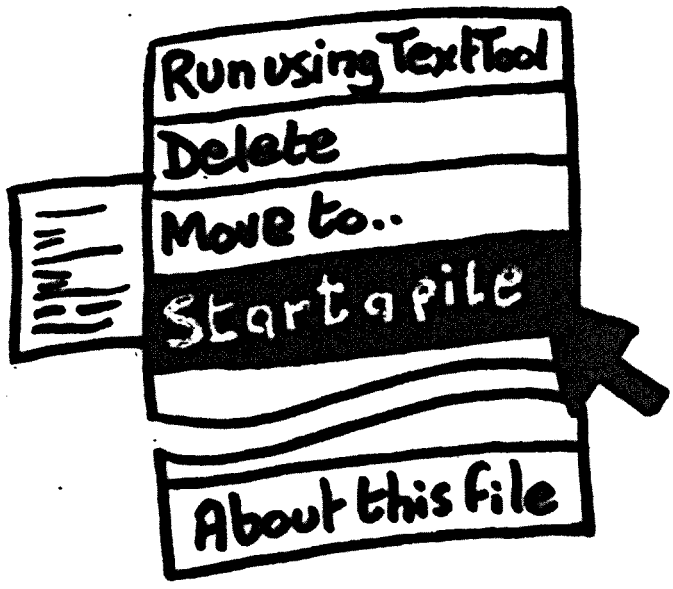


Figure 9.4 Starting a new pile

In a class-instance view of how categories of objects are realised, making a file the first element of a pile would suggest that the file would now multiply inherit properties and toolbar actions from a second pile category. In a prototype-centred view, new attributes and methods are simply dynamically added to the file's interface. Once a pile has been started, other files can be added to it. Drag and drop operations such as moving files into folders, or placing a file onto the top of a pile in Mander, Salomon, and Wong's (1992) prototype, risk semantic errors being made. The toolbar interaction style of Medusa is meant to prevent this possibility. A toolbar option that allows files to be added to an existing pile can be seen in Figure 9.5.

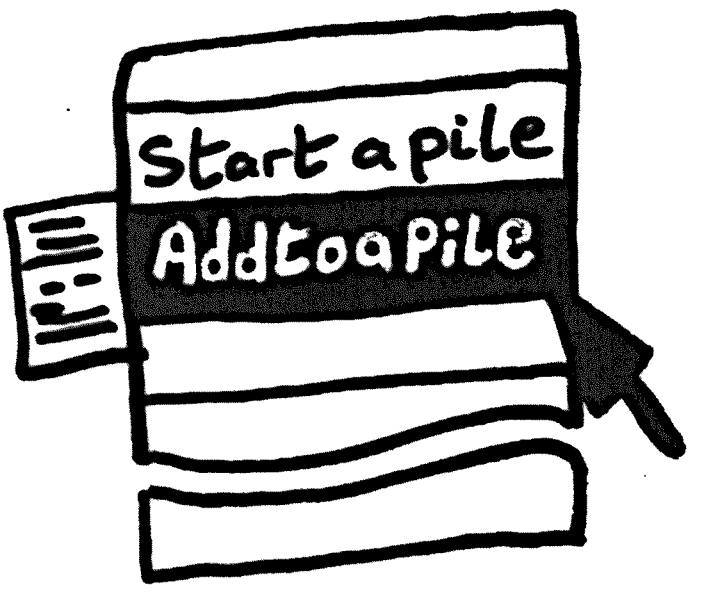


Figure 9.5 Adding a file to an existing pile

In a Mander, Salomon, and Wong's pile-centred system, the viewing cone is used to scan through the contents of a pile, in a pile-oriented version of Medusa a **Spread out contents** option would be placed on the toolbar but would only appear when the pointer is over a file that is a member of a pile. The pile is assumed to be the object towards which messages from the toolbar are directed, files must be removed from the pile if they are to be the focus of action, in keeping with everyday experience of piles of objects, and the containment schema underlying understanding of the pile. Spreading out the contents of a pile in a pile-oriented version of Medusa is storyboarded in Figure 9.6.

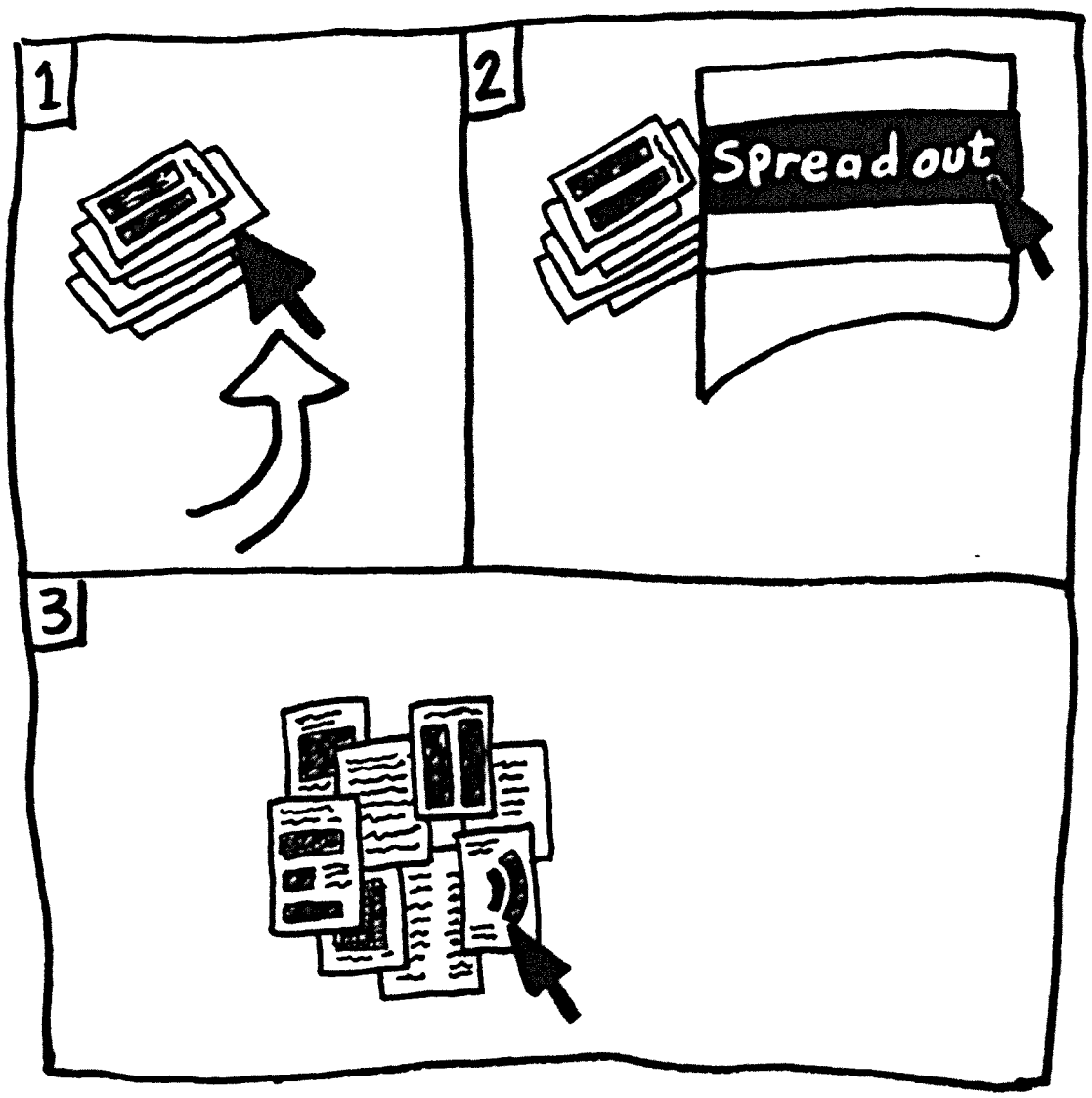


Figure 9.6 Spreading out a pile in a revised version of Medusa

How Many Piles can a File be in?

A pile is a depiction of all the files that meet the conditions of category membership that the pile denotes. Using the Lakoff/Johnson theory, it is claimed that understanding of the pile is based upon a PILE is CONTAINER metaphor, currently there is no formulation of machine support for pile creation in Medusa version two. In subsequent work, however, to provide such machine support, the risk of falling into the same trap as Lifestreams and the Semantic File System, described below, must be noted and avoided. Adopting the Aristotelean CATEGORY is

CONTAINER metaphor underlying the classical theory of category membership, according to Lakoff (1987) and Johnson (1987), must be avoided. Even without machine support to construct piles, the problem of determining which pile a file should be in remains. It is possible to argue that a file can be placed in many piles at the same time. In the folder metaphor this is achieved either by means of aliases or a filename-inode link, both of which were examined above. In the second version of Medusa there is only one copy of an object unless the file is explicitly duplicated by the user, reflecting experience of files in the physical world, our concern being to support direct manipulation as described by Lakoff (1987). If a file is duplicated, the problem of version control must be addressed, simple replication of a file's contents raises the problem of not only having to remember a file's location, but also the location of the version wanted, as can be seen in the protocols quoted in Appendix C.

Piles Across Volumes

The computer-computer metaphor makes explicit the presence of additional storage volumes connected to the workstation. The root window displays the piles supported by the internal hard disk, there is also, however, the problem of depicting the files stored on other volumes. In the folder metaphor and in the container concept of Medusa version one, the external volume is just another container and the interaction design of allowing *dragging* within directories and *moving* across directory and volume boundaries is adopted. There are no directories within a volume in the second version of Medusa hence allowing direct manipulation to occur. The design of existing operating systems makes the design of Medusa-style interfaces to this problem difficult.

For example, in UNIX file systems (each physical disk may store up to seven file systems) can be mounted into, and un-mounted from, an overall file space. Hiding

the implementation of filespaces is achieved in UNIX by forbidding links that cross file systems, the data and the links to it (what appears in a directory listing) must exist in the same file system. If a file system is un-mounted then its contents are invisible to both the user and system until remounted. In the Macintosh system, by contrast, aliases may cross volume boundaries and, as in UNIX, an un-mounted volume only becomes apparent by its absence, the volume's icon does not appear on-screen, and double-clicking on an alias's icon will cause an error message to be displayed. While neither the UNIX link nor the Macintosh alias owe their design to the LINK schema (Johnson, 1987: 117-119), the UNIX link is more in keeping with it, in that links between objects (filename/inode and datablocks in the case of UNIX) are typically "spatially contiguous within our perceptual field." Links between more than two objects and spatially and temporally discontinuous entities (action at a distance) are less typical. We have repeatedly stated that the difficulty with links arises at points of breakdown, either in breaking of the link itself, or in direct manipulation in the model world. In the second version of Medusa, physical world notions are adopted to improve understanding of the model world. For this reason links are not employed in the second version of Medusa, instead each icon depicts an instance of a file, if a file needs to be present in a number of piles then it must be duplicated. The problem that must then be confronted is that of version control, knowing and determining the state of an object in the model world.

A related problem, one also related to the problem of version control, is managing file organisation on volumes that are only occasionally connected to the workstation, such volumes including floppy disks, ZIP drives and equivalent removable disk technologies, and personal digital assistants with some storage capacity. The need to take such technologies into account means that it is not possible for Medusa to adopt a solution to managing the piles on the root window similar to that in the Kansas environment (Maloney and Smith, 1995). In Kansas, the root window is very large, and only a small user-selectable region of it can be seen at any one time. The

advantages of Kansas's being a synchronous shared workbench designed as a collaborative environment (although subject to the drawbacks inherited from the Alternate Reality Kit) are also lost in Medusa.

The same interaction style as in Medusa version one is employed in the second version of Medusa to implement copying files across volumes, copying being the default operation across the data path shown in the computer-computer metaphor. The toolbar contains **Move** and **Copy** operations and the user must specify one of the external volumes in the computer-computer metaphor on-screen schematic as the destination for the file or pile to be transferred. Dourish and Button (1998: 421), considering file copying in a system based on the folder metaphor, note:

> "... the abstraction that has been offered by the system — the folder — hides the details on which...understandings could be based. The differences between local and remote folders, the difference in the operation of local and remote copy operations, and the consequences of these differences are hidden from view.
>
> Furthermore, it is not sufficient simply to offer two different kinds of folders providing a distinction between local and remote ... Actions and accounts are situated within the specific circumstances of their production, not within abstract characterisations of them. In other words, what is important here is not the differences between two abstract types of copying (local copying and remote copying), but the specifics of this or that copying operation. There are far too many different features of the occasion (including distance, available network bandwidth, other people's activities, the types of files involved, and even the type of network infrastructure) for designers

or users to be able to distinguish among them in the abstract model that the system offers."

The second version of the Medusa system does not have local copying, only movement across the workbench. The problem of depicting remote copying remains, however. Dourish and Button, rather than "trying to provide different abstractions for all the different circumstances in which copying may take place", propose providing an *account* of copying, "a metaphorical frame drawn from the ethnomethodological perspective on the organisation of action." Dourish and Button's account of file copying is a schematic, shown in Figure 9.7, depicting data buckets and connections between them which is claimed to have some explanatory power in cases of breakdown.

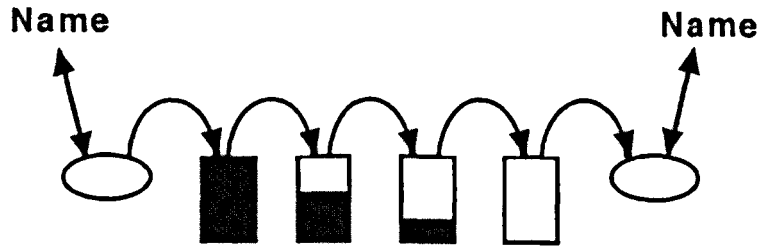


Figure 9.7 An account of file copying (Dourish and Button, 1998: 423)

Accounts are related to Dourish's (1995) notion of *reflection* in computing systems[2]. A reflective computing system is one in which a program has access to its own representation and execution environment, and is able to understand how a particular state came about and can alter its own subsequent execution. While notions similar to that of reflection form the Medusa system design, including object-based help and visibility and tangibility, Medusa is limited in the degree to which the end-user (and the system itself) can modify and re-program it. What Dourish and Button do not

[2] The idea of reflective computing systems is due to Brian Cantwell Smith, full details may be found in (Smith, 1996).

provide, it should be noted, is an appreciation or analysis of the metaphors underlying their account of file copying, and the functionality hidden, as well as that revealed, by their metaphors. It was stated in Chapter 6 that the computer-computer metaphor-inspired depiction of the copying process differs from Dourish and Button's account of copying. The relationship between the ideas underlying Medusa and reflective systems remains a topic requiring further consideration.

Versions

The ability to duplicate files, and to copy files onto disk volumes or PDAs that may belong to other users, raises the problem of version control and the depiction of different versions within both Medusa system designs. The problem is more acute, however, in the second Medusa system. There are a number of ways of managing versions of a particular file that the user may modify. One approach is to allow complete independence of objects, any modification produces a new version within the same pile or region of workbench as other previous versions. While possibly useful for some users, this approach permits the phenomenon quoted in Appendix C where the user may lose track of the location of the draft sought. Many operating systems limit the number of versions in the same container to the current draft and the most recent version prior to modification. The computer-computer metaphor schematic allows a change of version strategy to be included as part of the system behaviour that the user may control. Where duplicates of a file exist in a number of piles another version strategy that the user may adopt is for each copy to be a manifestation of the same most recent draft[3]. Two existing designs implement this functionality without the drawbacks of links and aliases. The first is the *publish-and-*

[3] An analogy may be found in particle physics, one answer offered to the question of why all observed and studied electrons have the same properties is that there is only one electron, but it moves around quickly.

subscribe model from the Macintosh operating system (described in Olsen, 1998, Chapter 13). In this model the user selects some information and informs the application that they wish to publish it. An edition file containing the published information is then created and the user must then create a subscription in the destination file. Any changes to the edition file cause corresponding changes in the subscription. Such a mechanism is compatible with the user determining how many past versions of a file are placed in a pile.

The publish and subscribe model is compatible with file movement across volume boundaries, and has advantages over the other mechanism for file synchronisation Olsen (1998) also describes, the *moniker*, found in Microsoft's OLE architecture. The first part of a moniker is an absolute pathname in which the linked information is stored, the second part are identifiers that reference the linked data within the duplicated file. In the moniker approach, the link may break and identifiers may also be deleted from the file during editing, making duplication and synchronisation even more difficult across distributed disk volumes. If a file is to appear in a number of piles, possibly on different disk volumes, a publish-and-subscribe mechanism will be required. Following Dix, Rodden, and Sommerville (1996), however, it is known that in a collaborative version of Medusa, with multiple disk volumes and occasionally attached PDA's, the notion of "the current version" of a file is almost meaningless, and that the Medusa system will need to reflect this. It is believed that little needs to be added to the Medusa system design to make this apparent, however.

Versions and Synchronisation in a Revised Medusa Design

File version and synchronisation mechanisms based on the idea of links, as described above, are prone to difficulties and the metaphor breaks down quickly. On PalmPilot devices, synchronisation of data when the palmtop device and the personal computer on which a duplicate of the data on the palmtop was once created (copies on either or

both machines may have since been modified) is performed by a *conduit.* In the Palm operating system, this term means a form of dynamic library that dynamically and temporarily extends the facilities offered by the personal computer's operating system in order to allow the synchronisation to occur. It is possible to investigate whether the use of the term "conduit" is more than just a case of designers needing to select or coin a term to name a particular type of computer program. Reddy (1993) proposes that our ideas of communication, and the language used to talk about language itself, are grounded in the CONDUIT metaphor. The components of this metaphor being:

> "(1) language functions like a conduit, transferring thoughts bodily from one person to another;
> (2) in writing and speaking, people insert their thoughts or feelings in the words;
> (3) words accomplish the transfer by containing the thoughts or feelings and conveying them to others; and
> (4) in listening or reading, people extract the thoughts and feelings once again from the words." (Reddy, 1993: 170)

Johnson's (1987: 59) list of the parts that make up the CONDUIT metaphor is more useful for considering the design of a file synchronisation mechanism that assumes his and Lakoff's theory of metaphor. Johnson's list of parts is:

> " 1. Ideas or thoughts are objects.
> 2. Words and sentences are containers for these objects.
> 3. Communication consists in finding the right word-container for your idea-object, sending this filled container along a conduit or through space to the hearer, who must then take the idea-object out of the word-container."

In the Lakoff/Johnson theory, metaphors are grounded in terms of image schemata. The CONDUIT metaphor is, in Johnson's description, grounded in a number of schemata. One of these is the COMPULSION schema (shown in Figure 9.8) in which a force has a magnitude, moves along a path and has a direction. In Figure 9.8, the solid line denotes an actual force vector, the broken line denotes a potential foce vector or trajectory. In the CONDUIT metaphor, the COMPULSION schema captures the illocutionary force of an utterance.

Figure 9.8 The COMPULSION schema (Johnson, 1987: 58)

The other schemata that ground the CONDUIT metaphor are BLOCKAGE, REMOVAL OF RESTRAINT, DIVERSION and COUNTERFORCE. The suggestion made in Appendix C is that the schemata that ground an interface metaphor possess entailments that must be addresses by attributes or actions provided to the user who must understand the metaphor and perform actions in keeping with the metaphor that change the state of on-screen objects. In a revised version of Medusa, the toolbar for the root window might contain the option **Create a Conduit**, as other object-based interfaces must allow the user to create new instances of objects, or must allow instances of them to be fetched from a convenient store. The issue of how a conduit object can be rendered will be ignored and will be left as a topic for further graphic design and usability testing effort. When a conduit object is created and becomes part of the Medusa model world. According to the Medusa design principles, it should be possible to interact with the conduit, it should possess an associated toolbar that will contain options that follow from the entailments of the schemata that ground the CONDUIT metaphor. A first list of suitable options can be seen in Figure 9.9. Further options might be added, but

they should, where possible, be entailments of the schemata that ground the CONDUIT metaphor.

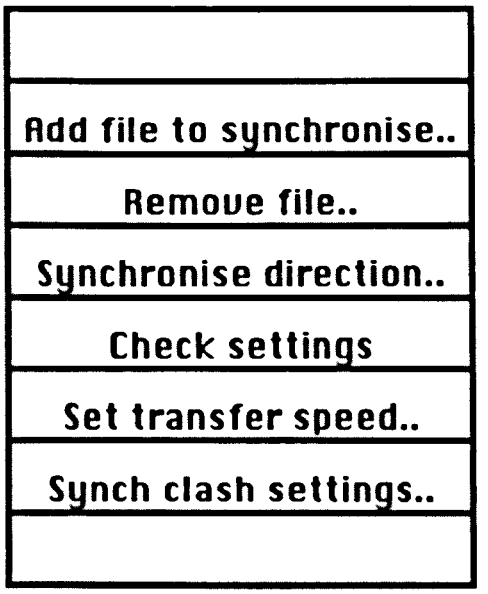


Figure 9.9 Toolbar options for a conduit

9.1.6 Other File Organization Solutions

Piles and folders are two important approaches to information organisation within interactive computing environments, important because they are prompted by existing practice of those engaged in tasks drawing on other information resources. Many have said, however, that the computer is not merely a tool, but also a system that can support new ways of working that are not merely imitated (or used as source domains) in user interface designs. Other user interface solutions to the problem of file organisation have been suggested, these are briefly surveyed below in order to determine whether other interface designs have advantages that suggest that any subsequent versions of the Medusa system should adopt these designs.

The Spatial Data Management System

While the Spatial Data Management System (SDMS) was first developed by Nicholas Negroponte, Richard Bolt, and their collaborators, in the 1970's, and examples such as Bolt's "Put that there" are well-known historical artefacts, the SDMS project can be claimed to be on-going as display and rendering technology advances. As Medusa is currently tied to the desktop, we are most interested in the early versions of SDMS, more recent versions being closer to notions of virtual reality environments. The first SDMS employed a wall-sized display upon which on-screen objects denoting items of interest could be placed and moved to meaningful locations. The SMDS system is claimed as being a major influence on the computer desktop, but computer desktops are smaller than real desktops, requiring additional metaphors such as Rooms or mechanisms such as Sun Microsystems' *workspace switch* in their Common Desktop Environment. SDMS, in particular in "Put That There", however suffer from some of the same problems that folders, piles, and environments in which both can be found, give rise to. While speech recognition and pointing de-referencing allow deitic reference to objects to be made ("put *that* there"), ambiguities such as that found by Mander, Salomon, and Wong (1992) still require resolution. If a folder is on top of a pile and an additional file is to be added to the pile, is the pile or folder the target? Section 6.4.5 addressed resolving this ambiguity of reference.

Dynamic Queries and the Semantic File System

An issue that is essentially at the heart of file organisation and interaction with information resources is category construction, "finding" information where file placement facilities support "reminding". Dynamic queries, an approach developed by Ben Shneiderman and his colleagues at the University of Maryland, of which FilmFinder is a representative system, combines direct manipulation with database

visualisation to allow users to filter information through the use of features such as buttons and sliders. Records in the database may be reduced to a manageable set by adjusting the range of values between which field values may lie using interface components that directly manipulate these ranges of values. While the user can quickly find records they are interested in, the dynamic query approach does not address how the records are described and indexed, categories (such as "thrillers" and "action movies"), and members of the categories are defined by those who build the database. For the user of a system in which they create many of the objects to be indexed and retrieved, and define categories to suit their tasks, dynamic queries offer little.

The semantic file system (Gifford, Jouvelot, Sheldon, and OToole, 1991) addresses tasks performed prior to those supported by dynamic queries. It constructs sets of potentially useful files, by giving additional semantics to files as well as providing associative access to a file system via virtual directories. Using familiar UNIX directory commands such as `ls` and `cd`, associative queries are interpreted to produce file listings that are more meaningful than the basic hierarchical directories that the semantic file system adds to. Transducers are devices added to the basic file system that associate additional attributes with each file extension (the filename's suffix such as ".C" which normally denotes the file's type). The mail transducer, for example, would associate the field-attribute pairs "from:", "to:", "subject:" and "text:" with each file with the suffix ".txt". It is recognised that a similar mechanism to the transducer is needed in an implementation of the Medusa systems in order to "register" new file types so that suitable icons can be constructed, among other functions. These attributes can be used in queries that resemble conventional commands to generate more meaningful lists of files. A query such as "`ls -F /sfs/owner:/smith`" described by Gifford et al. (1991) lists all the files owned by a user with the system name "smith" stored in the directory `/sfs`.

While adding much to basic file systems, the user interface to the semantic file system adds little to the UNIX command-based user interface, the failings of which have long been documented (Norman, 1981). Again the problem of command-based systems, forcing the user to have a conversation with an unseen agent about an unseen task domain, arise. While adding additional semantics to the file system, the extra attributes are hidden, as are the commands used to discover them, in addition the attributes depend on the person who constructs transducers. This person is not the user in existing implementations of the semantic file system. The types of objects that can exist within the file system are also still limited to the set of filename suffixes. As Medusarequires a form of object type registration mechanism, it can be hopefully seen that some of the more useful ideas introduced by the semantic file system can be adopted and improved upon.

Lifestreams

Lifestreams is prompted by a number of objections to the desktop metaphor, including an objection to the notion of the need to support location-based search mechanisms. Fertig, Freeman, and Gelernter (1996) note that Barreau and Nardi's (1995) studies of users of the Macintosh and a number of PC-based operating systems show the following similarities between users:

> "1. A preference for location-based search for finding files (in contrast to logical, text-based search);
> 2. The use of file placement as a critical *reminding function*;
> 3. The use of three types of information: *ephemeral, working* and *archived*;
> 4. The 'lack of importance' of archiving files."

Fertig et al. claim that similarities 1, 2, and 4 are artefacts of the computing systems studied rather than statements of the way users actually acquire, organise, and maintain information. Fertig, Freeman, and Gelernter's Lifestreams system proposes a new metaphor that replaces traditional files and directories. Lifestreams, shown in Figure 9.10, is claimed to be based upon the metaphor of a time-ordered stream of documents. Every document created is stored in the lifestream, the tail of the stream is the past, in the future the stream contains documents that the user will need, such as reminders, "to do" lists, and meeting schedules. In the present, the stream contains items such as work in progress and recently arrived e-mail. The claims made for Lifestreams include that the system supports reminding and archiving inherently in the model, and also that it aids in locating information. One way in which Lifestreams does this is by ephemeral and working information typically being located in the present part of the stream. The other is by allowing the easy creation and destruction of substreams by filtering the stream as a whole according to appropriate criteria.

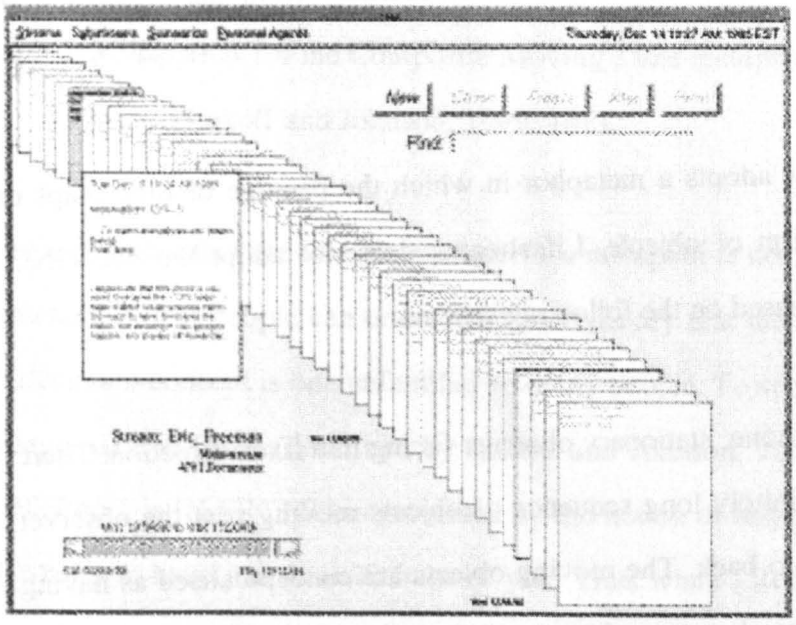


Figure 9.10 A Lifestream (taken from a video presented at CHI'96[4]).

[4] http://www.acm.org/sigs/sigchi/chi96/proceedings/videos/Fertig/etf.htm

As can be seen in Figure 9.10, the Lifestream of documents forms a diagonal line across the display. Documents in the present are shown in the bottom right-hand corner of the screen, documents are stacked so that the rear-most document in the top left-hand corner of the display is the oldest one rendered. After a period of time, documents "fall off" the edge of the screen and are automatically archived. A scrollbar allows the time parameter to be altered affecting the documents that are shown in the region of the display where "present" documents are displayed. In order to move into the future, however, so that reminders may be introduced, the Lifestreams system "clock" must be altered by a function reached from a menu option. Lifestreams is proposed as an alternative to the desktop metaphor, one that has the "organisational metaphor" of a time-ordered stream of documents. It is possible, however, to employ the Lakoff/Johnson theory to critique Lifestreams. The frequent use of the word "stream" to describe the Lifestreams interface is to employ an appealing metaphor. In terms of Lakoff and Johnson's (1999, Chapter 10) analysis of the metaphors that describe understanding of time, however, Lifestreams suffers from problems, and it may not differ considerably from the systems it seeks to replace.

While Lifestreams adopts a metaphor in which the passage of time maps onto the position and motion of objects, Lifestreams does not adopt the MOVING TIME metaphor, which based on the following schema:

> "There is a lone, stationary observer facing in a fixed direction. There is an indefinitely long sequence of objects moving past the observer from front to back. The moving objects are conceptualised as having fronts in their direction of motion." (Lakoff and Johnson, 1999: 141)

This schema combines with a TIME ORIENTATION metaphor, the mappings of which are shown in Table 9.2, to produce a set of composite mappings shown in Table 9.3.

The Location of the Observer	→	The Present
The Space in Front of the Observer	→	The Future
The Space Behind the Observer	→	The Past

Table 9.2 Mappings for the Time Orientation metaphor (Lakoff and Johnson, 1999: 140).

The Location of the Observer	→	The Present
The Space in Front of the Observer	→	The Future
The Space Behind the Observer	→	The Past
Objects	→	Times
The Motion of Objects Past the Observer	→	The "Passage" of Time

Table 9.3 Mappings for the Composite Moving Time metaphor (Lakoff and Johnson, 1999: 142).

It should not be concluded that the Lifestreams interface metaphor is not grounded in a pattern of interaction that people can understand easily, only that the schema that grounds the Lifestream concept is one unfamiliar to many people. To speakers of the Aymara language used in Chile (described by Lakoff and Johnson, 1999: 141) the metaphor "THE PAST is IN FRONT" is grounded by the notion of being able to see the results of what you have just done in front of you. Thus while Lifestreams may have an acceptable level of usability, its design conflicts with the culture and everyday experience of embodied interaction with the world of most of its intended user population.

One can also question whether Lifestreams is as radical an alternative to electronic support for the notions of piles and folders as it is claimed to be. The analysis of piles and folders presented in Appendix C shows that both these forms of file organisation can be understood in terms of the same image schemata and metaphors with similar mappings. Similar claims can be made for how Lifestreams is understood. The MOVING TIME metaphor that underlies Lifestreams (albeit combined with a mostly unfamiliar TIME ORIENTATION), , and the MOVING OBSERVER metaphor (the other mutually exclusive metaphor used in descriptions of temporal events in most languages) are both extensions of an EVENT-FOR-TIME metonymy (Lakoff and Johnson, 1999: 154). The example "The Kronos Quartet concert is approaching" given by Lakoff and Johnson (1999: 154) obtains its meaning by the event of the concert standing for the time of the concert, and the time is conceptualised as approaching. In the EVENT-FOR-TIME metonymy:

> "Times are then conceptualised as locations or bounded regions in space or as objects or substances that move. Events are then located with respect to those locations in space or objects that move." (Lakoff and Johnson, 1999: 155)

Thus within Lifestreams, newly edited or created documents and reminders are located with respect to locations or bounded regions in the part of the display denoting the stream. We could therefore undertake an analysis similar to that in Appendix C of how present, ephemeral, and sub-streamed documents are referred to and find that manipulation of items within the categories formed in these regions of time is reasoned about in the same way as a pile or folder. It is noticeable that, like the second version of Medusa, Lifestreams resolves the problem of ambiguity introduced by trying to integrate folders with the overall organisation structure of the user interface by not having a folder interface feature at all. Also noticeable are the

commands provided by Lifestreams to manipulate documents and streams, `new`, `clone`, `transfer`, `find`, and `summary`, most of which either have the same semantics as the generic commands, or those of class instance creation in object-oriented programming languages.

While Lifestreams may not be as radical an alternative to piles and folders as its creators believe, and relies for understanding on schemata that are unfamiliar to many users, the Lifestreams system does address a number of important problems. The most serious problems are the related issues of archiving and scalability. The pile metaphor does not address archiving, information is used only for comparatively short periods of time and then disposed of, suiting the habits of knowledge workers. Scalability is not a problem that needs to be addressed if the information to be employed in tasks or the creation of new files is ephemeral, but where archiving is employed the number of files in an information space may grow to be large. In Lifestreams, files that are pushed off the edge of the display are automatically archived. The user must scroll back into the past in order to enter the region of time where the (now invisible) files may be found, although most of the depiction of each file will be hidden by the more recent files. The user may also use the `find` facility, but this function serves to create a substream. It is worth investigation to determine whether time-based search (when did I create that file?) is as prone to the difficulties of location-based search (where did I leave that file?), especially in terms of the consequences mentioned above of conceptualisations arising from the EVENT-FOR-TIME metonymy.

Data Mountain

Data Mountain (Robertson et al., 1998) is a relatively simple means for storing and retrieving web documents. The Data Mountain, shown in Figure 9.11, is a texture-mapped rectangular plane segment angled at 60° to the horizontal plane extending

away from the viewer and rendered in perspective. The Data Mountain is intended to replace the "favourites" or "bookmarks" mechanism of world-wide web browsers as a means of noting and returning later to web pages of interest to the user. Thumbnail icons, reductions of a web page to icon size, may be placed on the mountain in locations meaningful to the user and icons may occlude others. If icons are placed at the top of the mountain the act of rendering them in perspective will make them appear smaller than icons placed at the foot of the mountain.

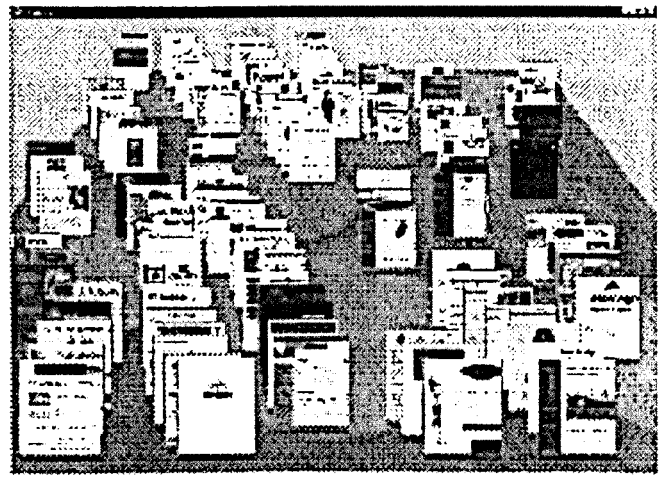


Figure 9.11 Data Mountain for web page favourites

(Robertson et al., 1998: 153).

Data Mountain, it should be noted, is, like the Spatial Data Management System and Perspective Wall, a *spatial* metaphor (Jones and Dumais, 1986), not a *spatialization* metaphor (Demasco, Newell, and Arnott, 1994; Regier, 1996) of the sort grounded in UP, IN, OUT, and so forth, schemas, discussed above. Lakoff and Johnson (1980) report that the mountain is a poor concept to ground in the body. Only reference to "the foot of the mountain", as we made in the last paragraph, is meaningful, and leads others (Bederson et al., 1996) to claim that the MOUNTAIN is BODY metaphor is a dead metaphor. Data Mountain is however interesting in allowing casual arrangement of icons collected together in space allowing ad hoc categorisation, and allowing spatial memory to aid in locating web pages sought.

This ability is very effective in the Data Mountain system (Czerwinski et al., 1999) even when the thumbnail images are removed leaving only blank icons, although mouse-over text giving the page's title for each icon is found to be required to maintain retrieval ability over long periods of time. There remain, however, unanswered questions as to the scalability of this design in domains other than web page bookmarks. , In a web favourites system the number of bookmarks is likely to be small and there is likely to be a fast turnover of those links that become redundant quickly (it is claimed that between 2 and 9% of any web search engine's indexed collection of links will be out of date at any time).

Alternative Interface Physics

The metaphor-based interfaces to file organisation systems considered so far have mostly been flat model worlds (ARK, the desktop, Rooms), or augmented real-world environments (DigitalDesk, metaDESK, and wearable computers). In these systems the image size is fixed and the user's viewpoint is changed, either by manipulating a set of 2.5D windows, moving to another Room, shifting the radar view, or by motion of the head. A recent alternative solution is to adopt a different physics in which the size of the workbench is fixed, often a screen-size in area, but the workbench can be deformed to bring regions of interest into the region of attention. According to Carpendale, Cowperthwaite, and Fracchia (1995: 219) this concept "... provides a useful metaphor for the actions performed to create the distortions. Pulling a section towards oneself to see it better, or ... magnify it, appears to be a natural response."

A system that implements alternative interface physics is the *Perspective Wall* (Mackinlay, Robertson, and Card, 1991). The Perspective Wall is a pliable flat sheet on which icons denoting documents are placed. The central region of the wall is placed closest to the observer's point of view, parallel to the plane of the screen, and contains most information in greatest detail. The portions of the wall either side of

the central region bend away from the observer and are rendered in perspective. If the user clicks on an icon on a portion of the wall outside of the central region, the wall scrolls and distorts until the icon lies within the central region and the portions of wall either side of the icon outside of the central region increase or decrease appropriately.

The Perspective Wall exploits computer animation technology, metaphors from pliable surface interfaces, and metaphors to the relative acuity of vision across the surface of the retina, but is still limited in the numbers and types of files it can provide access to. The version of the Perspective Wall described by Mackinlay, Robertson, and Card (1991) arranges icons by advancing creation date from the oldest to the youngest from the left of the wall to the right. The wall maps the file's type, drawn from a fixed set of categories, to the vertical dimension of the wall.

The PAD++ environment (Bederson et al., 1996) has a more pliable surface than the Perspective Wall. It may be deformed by the semantic zooming process on a more local scale within the central region, which in the case of PAD++ occupies the entire screen. The regions outside the central region can be brought into view using links and portals to other parts of the model world. PAD++ has been employed as a framework in which a number of familiar types of systems have been implemented, these include a world-wide web browser and a file directory browser. The browser uses thumbnail icons of web pages, as in the Data Mountain, to depict particular pages. These icons can be enlarged by zooming to better determine the identity of the web document, or completely enlarged until the document is rendered full-size and as the focus of the user's attention. The directory browser is similar to the PAD++ web browser, but files are only depicted as coloured squares, the colour denoting category membership, until zoomed to the maximum magnification possible where the file's contents become visible.

Further Developments of the Pile Metaphor

The initial work on the pile metaphor undertaken by Mander, Salomon, and Wong (1992) was based upon prototypes built using Macromedia Director. These prototypes, however, were *shallow*, in that most proposed user interface features were implemented but not linked to any underlying functionality. Subsequent prototypes employed clustering techniques to support both piles and documents in the same information space and both direct manipulation and automatic sorting tasks. Using clustering to collect together seemingly related files allows the task of subpiling to be delegated to an electronic assistant. How the 'agent' that performs this task can be comprehended as part of a desktop metaphor (viewing it as part of a wider OFFICE TASKS metaphor raises the problems encountered with Ed discussed in Chapter 4) is not explained though. Unfortunately work (Rose et al., 1993) that might have continued on employing clustering techniques to widen the sorts of categories of objects that piles contain; to allow user-defined categories to be automatically constructed from more complex file attributes; and to resolve remaining pile user interface's inconsistencies, was abandoned (Rose, 1998) when Apple's Advanced Technology Group was disbanded. It is clear, however, that work picking up where on piles left off should be added to the topics for further study if the scalability problem of managing files in user interfaces to organisation mechanisms is to be addressed.

9.2 Medusa-τ: A System Addressing Temporal Problems

In their view of design, taking into account long-standing criticisms of the Objectivist tradition underlying classical views of metaphor, Winograd and Flores (1986: 178) state:

> "Computers have a particularly powerful impact, because they are machines for acting in a language. In using them we engage in a discourse generated within the distinctions set down by their programmers. The objects, properties, and acts we can distinguish and perform are organised according to a particular background and pre-understanding. In most cases this pre-understanding reflects the rationalistic tradition we have criticised ... It includes biases about objectivity, about the nature of 'facts' (or 'data' or 'information') and their origin and about the role of the individual interacting with the computer.
>
> We have argued that tools based on this pre-understanding will lead to important kinds of breakdown in their use."

It has already been discussed above how the mismatch between the objects and objects' attributes, and tasks that objects' behaviours support, provided in a model world and what is suggested by the metaphorical source domain can cause breakdowns in user interfaces. Delays and lags are other causes of breakdown in metaphors and analogies in model worlds. The tradition that informs Winograd and Flores suggests that breakdowns force the user into a state of having to account for the breakdown, a true state of *being in the (model) world* is denied them. Rather than being able to naturally perform their work in the task domain, additional effort must be expended to model the system. In addition to the sources of breakdown caused by failure of metaphors, breakdowns also occur due to the unpredictable temporal behaviour of computing systems (Dix, 1987).

Two approaches may be found to solve the problem of temporal behaviour of systems that users may find troublesome. One approach is a design solution, the temporal behaviour of the system may be examined, modelled and understood (both

from a system and the users' perspective) and the user interface may be designed to incorporate features which explain the temporal behaviour within the system's conceptual model. This approach is adopted in the design of the first two Medusa systems. Another approach is a technological approach. The system may be designed and implemented so that the temporal behaviour of the system is controlled and guaranteed. An example is the attempt to guarantee the instantaneous response to typed input in a commercial word processor by only updating the line on which the cursor currently lies, and updating the layout of the remainder of the document visible on-screen when the user pauses typing (reported in Dix, 1991).

The Medusa-τ system is based upon the Medusa system, and assumes the same task domain as the Medusa system. The concern of the Medusa-τ system is to guarantee, where possible, the temporal behaviour of the system so that feedback of the system state and updating of the diagrammatic display depicting parts of the system image are appropriate and immediate. This approach simplifies the conceptual model and the interface design, the details of software design, however, as well as the implementation details, become more complicated. The issues of software architecture, implementation, operating system design and treatment and modelling of concurency and real-time system development require considerable attention if the Medusa-τ system is to be successfully implemented.

Some authors doubt that the real-time behaviour of systems needs to be considered. Took (1990a: 126), for example, claims that:

> "... timing is much less critical in general user interface systems than in process control applications, for example, because human users are more tolerant of delays or variations in timing than machines."

Hill (1992) states his hope that the complexities of real-time programming can be avoided in user interface design. The need to consider temporal issues in user interface design is becoming more widely recognised (Johnson and Gray, 1995), however. When polled as to their requirements, users state that they prefer fast system responses, and data exist on the optimum rates for displaying text, for example. Feedback can, however, be too immediate, if information flow is from the system to the user, there are maximum rates at which information may be presented if users are to be able to obtain information from the display (Card, Moran, and Newell, 1983).

Where the effects of actions and operators on the state of the user interface must be interpreted or learned for later use, reinforcement in the learner's mind that a particular action helps bring about a desired system state is reduced if the system feedback, and subsequent reinforcement is delayed (Kaelbling, 1993). Effects of system response time on the strategy users employ when interacting with user interfaces have also been noted. With increasing delays between user input and system response, users avoid actions which may cause errors and do not request output to confirm the system state, actions are increasingly planned and experimentation avoided (Grossberg et al., 1976). Where system delays are shorter than the many seconds in Grossberg's et al. (1976) experiment, but also vary, changes in user strategy have been observed by Teal and Rudnicky (1992) (although not as clearly in the replication of Teal and Rudnicky's experiments performed by O'Donnell and Draper, 1995). Their experiment considered user input to an unbuffered system where delays between user input and system feedback varied, and users were unable to enter further input until feedback was received. Where delays exceed 1.75 seconds, users are said to adopt a monitoring strategy, waiting for system feedback before continuing. Between delays of 0.75 seconds and 1.75 seconds users are seen to adopt a pacing strategy, entering input to the system without waiting for feedback, judging delays between input and adjusting the delay

in response to errors where the judged delay is shorter than the actual system delay. Where delays are shorter than 0.75 seconds the user is able to adopt a strategy of automatic performance, latencies between keystrokes due to cognition and the human motor system are longer than system delays.

As well as long delays between user input and system feedback causing users to alter their behaviour to compensate, and the learning of systems being complicated, a number of other arguments for addressing the temporal behaviour of user interface software have been proposed. The seven stage model of interaction with interactive systems proposed by Norman (1984), for example, relies on there being no perceivable delays between the execution of user actions and system feedback for the loop of interaction to be maintained.

9.2.1 Implementing Medusa-τ

Work has progressed on the refinement and implementation of systems described in the Agent notation (Treglown, 1998), and we have now a complete formal operational semantics and heuristics for converting the required external behaviour of agents into Java code (but not a full set of refinement laws). Methods for converting a timed Agent model, developed for the task of describing systems such as Medusa-τ into a suitable language running on a suitable combination of hardware and operating system, remain to be completed.

9.3 Conclusions

The first version of the Medusa system, the version that has received most design effort, while attempting to overcome known difficulties in the application of metaphors in the design of the model world is based in the Objectivist tradition rejected in the last chapter. In this chapter, two revised versions of the Medusa

system, one taking into account the Lakoff/Johnson theory of metaphor understanding, another recognising breakdowns introduced by the temporal behaviour of interactive systems, were discussed. In the following chapter, the contributions of the Medusa systems to HCI, and suggestions for further work, are presented, as are conclusions drawn from the work presented above.

Chapter 10

Conclusions and Further Work

What keeps you awake at night?

Trying to finish the phrase 'A bad simile is like a...'

— Mark Lamarr, questionnaire column, *The Guardian*, 5/9/98.

10.1 Summary of the Thesis

In this chapter we summarise the work undertaken, suggest the contribution of the work to HCI theory and to user interface design, indicate further work immediately arising from the work reported on here, and introduce on-going work to resolve unfinished problems. The work undertaken and reported in this thesis can be summarised as follows:

- a survey of interface styles and design methods revealed the problem of choice in the user interface design process, and resulted in the role of metaphor and analogy in user interfaces and in user interface design being considered for investigation,
- a survey of a number of important and influential systems that are based on explicit metaphors in their model worlds revealed the important concepts, interaction styles, and widgets that these systems introduce,
- a literature review showed the limitations of employing metaphors and analogies in previous user interface designs,

- a small-scale study of first-time Apple Macintosh users supported findings of a previous similar study and identified problematic features for which improved user interface designs were judged to be needed,
- a literature review sought to examine where metaphors fit into the wider context of users' mental models of interactive systems and the pervasive nature of metaphor in these models was discussed,
- an analysis and model-building exercise of current systems applied the QPT method of describing mental models which is not usually employed in HCI. These models revealed inherent flaws in some metaphor-based systems, and revealed that some reconsideration was required of the domains between which metaphorical mappings are thought to be made. This consideration supported Laurel's (1993) previously proposed analysis of domains and mapping in interface metaphors,
- an examination of current system design showed the failings of existing user interface metaphors and of the existing theories of metaphor understanding,
- an application of a contemporary theory of metaphor due to Lakoff and Johnson to user interface design was made by recognising the need to be aware of results in formal semantics that question the nature of metaphorical understanding previously assumed in HCI,
- a number of case studies extended the limited use to which the Lakoff/Johnson account of metaphor understanding has previously been put in analysing user interface designs, and which further demonstrated the usefulness of this approach,
- a new system design named Medusa based on the guidelines and the QPT models discussed in Chapter 5 was designed,
- revision of the first Medusa system design were presented based on the results of usability testing, and based on the results of applying the theory introduced in Chapter 8 as a generative source of novel user interface designs,
- a comparison between the second, revised, design and recent user interface designs for the same task domain showed the comparability of these designs and suggested optimism for the usability of an implementation of the revised system design.

10.2 Contributions of the Thesis

Bannon and Bødker (1991) identify two approaches to design within HCI, a *task analysis* approach, which they criticise and reject in favour of an *artifact* approach. The task analysis approach informs the development of the first Medusa system. This approach is characterised by the assumptions underlying cognitive science and psychology, and the use of task analysis methods and programmable user models in design. In this approach, the computing system is programmed from an analysis and structured description of the tasks currently performed by eventual users of the new system, or from an analysis of the cognitive resources and knowledge structures needed to perform the task.

The artifact approach, by contrast, assumes that tools (hence also computing systems) are only fully revealed and understood *in use*, where "in use" has a far wider meaning than studying systems in the laboratory with representative users as subjects. While the second version of Medusa cannot claim to be informed by the artefact approach, it is based on assumptions that criticise some of the assumptions underlying the first Medusa system. Below we discuss the contributions of the two Medusa systems in terms of the two approaches to HCI design, and also in terms of one set of suggestions for key HCI issues that should be addressed.

10.2.1 Medusa in a Cognitivist Framework

The first Medusa system, described in Chapter 6 and evaluated in Chapter 7, is grounded in the traditional cognitivist framework that is rejected in the methods of analysis used in the design of the second Medusa system. The use of the Qualitative Process Theory in Chapter 5 to describe mental models of model worlds based on physical world metaphors, and to provide a semantics to user operations in order to

explain changes to objects in the Medusa model world assumes an Objectivist world view. The work on Medusa version one makes two contributions. Firstly, Laurel's (1993) notion of user interface similes is strengthened as identifying the important domains between which mappings should be made in user interface metaphors. Secondly, mismatches between the on-screen model world and the underlying functionality are identified as a key source of user difficulties in understanding the system.

In addition, the way of viewing software that this contribution employs has also been acknowledged[1] as being an influence on the design of the Ontological Sketch Model (Blandford and Green, 1997) for modelling user interfaces and identifying usability faults. The Ontological Sketch Model (OSM), as its name suggests, requires the system designer or analyst to construct an ontology of interface objects and the actions that can be performed on them. The analyst lists the things that the user must know about in the interface, their attributes, accessibility, relevance to either the application domain or the device domain, whether the object is visible and whether or not it has a meaningful name or symbol. OSM, being more of a system engineering approach, captures aspects of the model world that QPT does not, QPT not being initially devised for use in HCI. OSM and QPT are comparable, however, in the number of aspects of system described in models of user-initiated actions and the effects that these actions have on interface objects. QPT describes these effects in a more formal way, however, and more tools exist that currently do for OSM to make predictions of the outcomes of effects. A section of an OSM description of a drawing package (taken from Blandford and Green, 1997) can be seen in Table 10.1.

---

[1] Thomas Green (personal communication) 14th November 1997, seminar at the Knowledge Media Institute, The Open University, Milton Keynes.

Action	Object	Effect	Context	Notes
click	drawing area	lay down a point for a sketchy-line	discrete mode	
drag	drawing area	lay down shape for a sketchy-line	continuous mode	speed of dragging affects sketchiness

Table 10.1 Part of an OSM table for a drawing package

10.2.2 Medusa in a Cognitive Semantics Framework

The second version of Medusa adopts the Lakoff/Johnson contemporary theory of metaphor in order to account for how some interactive computing systems can be understood. While the application of this theory to interactive computing systems is not unique, our application of it began independently of Rohrer's work (1995). The contribution of the work contained in Chapters 8 and 9 and in (Treglown, 1999) is to demonstrate the applicability of the Lakoff/Johnson theory to describe a wider range of interactive systems than it has been attempted to describe before. This work also promotes the Lakoff/Johnson contemporary theory as a candidate theory applicable to the design and evaluation of computing systems. The comparative analysis presented in Chapter 9 and Appendix C shows the value of applying the contemporary theory of metaphor as a predictive and analytical tool to reason about modern metaphor-based software technology. Benyon and Imaz (1999) demonstrate their recent adoption of the approach to design suggested by employing the Lakoff/Johnson theory in HCI and show that the contemporary theory is gaining the attention of other members of the HCI community.

10.3 Does the Work Address Key Issues in HCI?

Shneiderman (1986) identifies seven key issues that HCI should address, in this section we examine whether the work conducted and reported in this thesis is appropriate in terms of work that is deemed valuable and necessary, and whether the work addresses any of Shneiderman's challenges to researchers. According to Shneiderman's (1989) more recent, but less finely delineated, identification of important future directions in HCI research, we can claim that the work undertaken addresses the need to cater for office practice and the inclusion of more complex documents (containing media other than just text) in the model world. We can also claim that the Medusa systems contribute to understanding the temporal behaviour of interactive systems. Below we consider in further detail the contributions of the work in terms of Shneiderman's (1986) classification. While the detailed challenges he sets the HCI community are presented in terms of the prevailing technologies of the era in which his paper was published, the delineation of research problems is still valuable.

10.3.1 Interaction Styles - What is Natural?

In Shneiderman's analysis, natural interaction is said to be strongly related to the notion of directness, irrespective of the modality of interaction. Frolich (1993) observes that the meaning of directness has altered from Shneiderman's original meaning of the "first personness of interaction through manipulation" to Hutchins, Hollan, and Norman's (1986) meaning of it being a combination of distance and engagement. Engagement refers to *first personness*, the sense that the on-screen objects *are* the actual objects being manipulated. The term *distance* is employed in Hutchins' et al. conception of direct manipulation systems to refer to the complexity of mapping goals to actions meaningful to the computer at the interface. Systems

termed direct are designed so that this distance is minimised. Frolich notes that it is legitimate to apply the notion of directness to both conversational and manipulative systems and that the trend towards interfaces exclusively based upon manipulative interaction is an accident of history and is a trend that he states should be halted. For tasks such as information retrieval, mail handling, time handling, and programming, conversational and mixed mode systems are said to be more appropriate and more direct than action-based systems.

Frolich also claims that the historical association of directness with model world interfaces leads to an assumption that the use of real world metaphors improves directness. He suggests instead that it is also possible to conceive of direct conversational systems which do not employ metaphorical devices to reduce psychological distance, and also that the traditional historical association diverts attention away from supporting action-based interaction by using non-metaphorical icons to represent abstract computational structures. While we agree with another of Frolich's observations, that some real-world metaphors can result in indirect systems that do not enhance the user's experience of using the system, we offer different views to Frolich's. In the light of modern theories of metaphor examined and employed in this thesis we disagree with Frolich's central claim that designers should be encouraged to be sceptical about choosing action-based solutions to design problems. We also disagree with the suggestion that visual formalisms (that are claimed to rely less on metaphor for their semantics) and that language-based or mixed language/action-based forms of interaction should be used.

The contemporary theory of metaphor suggests that language cannot be as free of metaphor as Frolich believes, and that metaphor is central to cognition and semantics. Neither is it clear that, in the light of the contemporary theory, that visual formalisms, as described by Nardi and Zarmer (1993), are entirely free of metaphor in their semantics, as has been suggested. Graphs (x-y plots of data), for example,

are often mentioned as examples of visual formalisms, but, as is employed in some audio representations of data values using varying pitch (Buxton et al., 1985), underlying the semantics of the representation is an *up is more* metaphor. In other visual formalisms, for example graphs (nodes and links) such as Petri nets, quantities such as time may be grounded in terms of physical location in the diagram. Where visual formalisms are also dynamic and interactive, with further investigation it may prove that metaphors, in the terms that we now think of them, may be relied upon more for understanding of the formalism. than claimed.

In contrast to Frolich, we claim that while some metaphors can produce indirect systems, the key to directness is not necessarily to employ language-based or collaborative manipulation interfaces (described below). Instead we claim that directness is a product of the type and complexity of image schemata that ground a metaphorical mapping. We have shown that a feature found in an implementation of the desktop user interface breaks the invariance principle and is hard to account for. It also fails to suggest suitable actions that would allow tasks to be performed. The schemas that ground interaction with the second version of the Medusa system are simple. The resulting interaction with the system would appear to be direct. A claim that we tentatively propose, and shall investigate as further work, is that directness is a concept related to Lakoff's invariance principle, and that collaborative manipulation systems should be adopted only when the schemata that ground metaphors and actions become complex or the invariance principle is broken. The Medusa systems, and the approaches adopted to understand them, do, at least, seem to provide a framework in which directness can be consistently discussed.

10.3.2 Input Techniques - Putting Intention into Action

The choice and use of particular input devices with an interactive system is a topic related to the issue of naturalness of interaction. It remains a topic of ongoing work

as to how the amount of distance and directness between user intention and system terminology is changed by the use of different input devices. Application of results reported in Jeannerod (1997) suggests that a number of recent and novel input devices can be very direct if used in physical world metaphors. The mouse, however, despite its prevalence as an input device in direct manipulation interfaces, presents a number of problems. We cannot yet detail the image schemata underlying spatiomimesis and mouse-based interaction in general, and account for the reduced directness that the mouse seems to give rise to. Work to fully provide a rigorous grounding of mouse-based interaction in terms of image schemas and metaphor is ongoing. This work is likely to draw on results discussed in Lakoff and Johnson (1999). This work demonstrates that where Regier (1996) shows how the linguistic and pre-linguistic spatial concepts that form many of the image schemata that we have employed in the analyses of user interfaces above can be acquired from prototypical examples, these spatial concepts can also be used to suggest and generate suitable motor skills to perform tasks (Bailey, Feldman, Narayanan, and Lakoff, 1997; Narayanan, 1997).

10.3.3 Output Organisation

Concern for visibility and tangibility ensures Shneiderman's recommendation to enforce consistency in the model world. The use of either the browser metaphor or pile metaphor for file organisation reflects the user's need for organisation (either messy or tidy) and classification of objects according to the immediate needs of their tasks and their category structures, according to the version of Medusa being used. The more focused concerns under this issue, such as the fonts and colours used, remain design options to be addressed if a full implementation of the system is developed.

10.3.4 Response Time

The need to account for the temporal behaviour of the Medusa systems was mentioned above alongside descriptions of the user interface features in Medusa that are intended to aid the user in forming a useful and accurate mental model of the system. We have suggested throughout the thesis that the designs of both versions of Medusa, and the design of Medusa-τ, are motivated in part by the need to address the issue of system response time, and to account for a system's temporal behaviour. The Medusa systems and Medusa-τ adopt two different strategies in their design, respectively providing the user of an account of the cause of temporal breakdowns, and attempting to ensure that breakdowns do not occur.

10.3.5 Error Handling - Preventing User Errors

It is normally assumed that in "extreme" direct manipulation interfaces, i.e. those that implement physical world metaphors, it is not possible to make errors, as commonly understood. Alternatively it is assumed that only semantic errors can be made, where the user is not prevented from performing erroneous physical actions that have little sensible meaning in the machine's terms. In the first version of the Medusa system, the object-message style of interaction limits the number of errors that can be made; messages that cannot be sent to an object in its current state do not appear on the toolbar. In the second version of Medusa tasks that might cause errors to arise, particularly file movement tasks, are less likely to occur as the semantic distance between the on-screen world and the system semantics is reduced. Users may still perform tasks that they may not actually have wanted to perform, however, and will require undo and recovery facilities.

As noted in Chapter 7, the first Medusa system makes no explicit provision for undo and recovery facilities within the model world and in the support of users' tasks.

Undo (the provision of a feature that allows a previous system state to be returned to), and recovery (the ability to return to a previous state and to rerun history issuing a different set of commands), present particular difficulties in the design of an interactive system. Neither facility is product-oriented (task analysis cannot fully reveal the ways in which recovery might be conducted using a facility yet to be introduced). Rather the true usability of such features will be revealed in system use, the tool itself will be changed by the introduction of an undo facility, and so the true nature of interaction with a system that supports undo cannot be fully predicted. An analysis of the schema that might underlie potential metaphors for undoing commands remains to be conducted. This is despite the need to provide one for the Medusa system identified in Chapter 7, and the recognised need to understand the limits and possibilities of undo in metaphor-based user interfaces in general (Tognazzini, 1992: Chapter 10).

10.3.6 Individual Differences

The differences of gender, age, ethnic background, cultural heritage, and so on, that Shneiderman judges must be accounted for by design guidelines are not considered in any detail in this research. The intended user population of the Medusa systems is all users of the systems who perform tasks supported by the operating system through necessity not choice. Therefore, it should be possible to perform all tasks supported by Medusa with little expertise. The issue of cultural diversity and metaphor understanding in interactive system design is briefly discussed in Section 10.4.2.

A topic that Marcus (1993) discusses, that is also deserving of further investigation continuing the work begun and reported in this thesis, is the need to be aware of cultural diversity in the design of computing systems. In Marcus' analysis, the types of metaphor that he considers important in forming the basis for a user interface

design, and the types of metaphor that he feels help the designer to design a product for an international audience, are metaphors that are broad in scope and that are meant to encourage understanding of a large part of the system's functionality. These metaphors are subject to the problems described in Chapter 3 and by Johnson (1994). The use of metaphor in the second version of Medusa described in Chapter 9 recognises the centrality of metaphor in cognition and understanding claimed by the Lakoff/Johnson theory of metaphor understanding. In the contemporary theory, cultures define the categories that people possess, their conceptual structures, and the prototypical effects in category usage that will arise from the categories and conceptual structures. One conclusion that the contemporary theory allows us to reach is that since cultural effects are demonstrated even in the very basic image schemata that underlie people's understanding of the world, the strictly action-based Medusa version two system, or any other direct manipulation interface, can never be an "interface for all." Lakoff's (1987) survey claims, for example, that even FRONT-BACK schemata (very basic and common patterns of interaction with the external world) differ across cultures. However there exists within Medusa considerable scope for exploring further limits to the comprehensibility and usability of direct manipulation interfaces where metaphor is employed in their meaning and understanding.

10.3.7 Explanatory and Predictive Theories

Shneiderman's (1986) most firmly stated demand is for HCI to develop robust theoretical foundations, theories forming a basis for research, design guidelines, and teaching. The complaint underlying the research reported here is that while metaphor is thought to be central to understanding the world (and user interfaces), and while theories of metaphor and metaphor understanding exist, few applications of specific theories of metaphor as applied to HCI seem to exist. The work reported here has sought to remedy this, and as such can be said to address one of the issues that

Shneiderman judges important. The work undertaken to date has not invalidated the notion that the Lakoff/Johnson contemporary theory is a promising candidate theory capable of accounting for much of metaphorical understanding of interfaces, and understanding of metaphor-based systems. A continued investigation of how successful and useful a predictive tool the contemporary theory can be in user interface design is ongoing, and some discussion of this work can be found below. In some views of the process of scientific endeavour, a theory is of worth if it is, in principle, falsifiable. Further work using the Lakoff/Johnson theory should therefore address criticisms of it (for example Vervaeke and Green, 1997).

10.4 Suggestions for Further Work

10.4.1 Full Implementation of the Medusa Systems

As discussed in Chapter 7, usability evaluation methods are based upon assumptions about the nature of learning and using interactive systems. These assumptions determine the types of usability errors revealed. Usability evaluation methods also differ in the *number* of usability errors that they reveal. The cognitive walkthrough method employed in Chapter 7 reveals only a small number of usability errors, this number would be increased if the number of system evaluators were increased. The use of a usability inspection method was required due to the lack of a working prototype of the Medusa system described in Chapter 6.

The Medusa systems are grounded on particular models and theories, and a partial implementation of the first version was based on an (again partial) formal specification using the Agent notation (Abowd, 1990). The aims of the research include examining the possibility of specifying aspects of direct manipulation. The research aims also include examining the possibility of being able to formally describe system features which can maintain metaphors that do not suffer from the

breakdowns common to existing metaphors. However, even if a full specification had been completed, and principles of usability, such as those provided by Dix (1991), had been applied to the specification and the specification verified, it is unlikely that all usability errors would have been revealed. Work by Harrison, Roast, and Wright (1989) and Wright, Merriam, and Fields (1998) shows that the abstract model of a system cannot reveal all usability problems; testing involving a completed system and human subjects is required to reveal the true range of usability problems. Given the more complex model of metaphor and cognition assumed in the design of the second version of the Medusa system, the need for user testing is even greater than for the first Medusa system. The "scientific" model of usability analysis presented by Wright, Merriam, and Fields (1998) is of particular interest. For example, the testing of claims made by formal models by usability testing allows formal models to be refined and made more useful. The empirical study reported in Chapter 4, and the issues surrounding metaphor, action, and categorisation demonstrated by recent models and theories show that data obtained from people is the most valuable source of data when seeking to understand the user's understanding of an interactive system. Progress toward identifying useful interface metaphors and the schemata that structure them can be made by investigating existing systems and tasks in detail. Given a suitable corpus of interaction data, existing metaphor-based design methods seek to determine the verbs and nouns making up the task domain. In our approach, demonstrated in Appendix C, we attempt to identify the spatialisation, and other, metaphors that the task domain is understood in terms of. It is these larger metaphors that we will use in future to generate better visual representations and interaction sequences.

10.4.2 Implementing Agents

The formal notation explored as a means of specifying the Medusa systems, as mentioned, was Abowd's (1990) Agent model and language. While work has

continued on applying this model to the specification of interactive systems, and the problem of refining the specification into code (Treglown, 1998), refinement into all programming languages suitable for interactive user interfaces is not yet possible. Also, the additional theory developed, while permitting refinement of the external, dynamic, behaviour of agents, relies on transformation by hand; tools to automate the refinement process remain a topic for further work. The agent model is, however, unsuitable for specifying some classes of systems that work in user interface metaphor must address. The model of concurrency adopted by the external behaviour of all the agent models that we have employed to date in this work cannot capture truly concurrent events. This problem will need further attention if the problem of designing a collaborative and shared Medusa environment is considered. The first step in this process is to continue work reported in (Treglown, 1998), and to generate a semantic equivalence between agent specifications and modules of code in a suitable programming language. This should be done instead of continuing to translate a transition system compiled from the external behaviour components of the agents making up a system into high-level language code in a principled, but informal way.

10.5 The Future of Metaphors and Direct Manipulation

10.5.1 Classes of Metaphor and Understanding Directness

The types, or paradigms, of interaction that the user might have with a computing system are defined by Hutchins (1989) who refers to these types of interaction as *interaction metaphors*. The definitions employed in this thesis are Hutchins', but we have avoided referring to them as metaphors to reduce confusion. The types of systems we considered in this work were said to be based on the model world paradigm, in contrast with the conversation paradigm. Hutchins defines two other interaction metaphors, however, and some consideration should be paid to these,

especially if they should become increasingly relevant, and prevalent, in future systems. The declaration metaphor is based on ideas from speech act theory where utterances are sufficient to change the state of the world (for example, "I pronounce you man and wife"). A declarative interface differs from the conversation paradigm in that utterances have a "causal force" in the world. The declaration metaphor is a poor metaphor, however, as when the user issues an ungrammatical expression, no change in the world occurs. Thus if the user issues an expression with no causal force, or one that cannot bring about a change in the state of the world, there is no way to filter out or report objections to these expressions. Interaction with such a system would eventually prove frustrating, not least because many expressions, notably those involving deictic reference, cannot be acted upon.

The final interaction metaphor defined by Hutchins (1989) is *collaborative manipulation*, which is depicted in Figure 10.1. In this interaction metaphor, Hutchins states that the computer should be an actor in the setting in which it is employed and thus should behave as a human does in human-human interaction and should support conversational interaction. The model world metaphor, the interaction style that has been the subject of this thesis, is based on the assumption that people are skilled at manipulating objects in the environment. Because work is often conducted in a social world, and in a collaborative manner, this implies that user interfaces should consist of both a model world and an intelligent agent. Both the user and the agent should have equal ability to alter the system's state, but the agent can automate those tasks that are tedious for the user to perform.

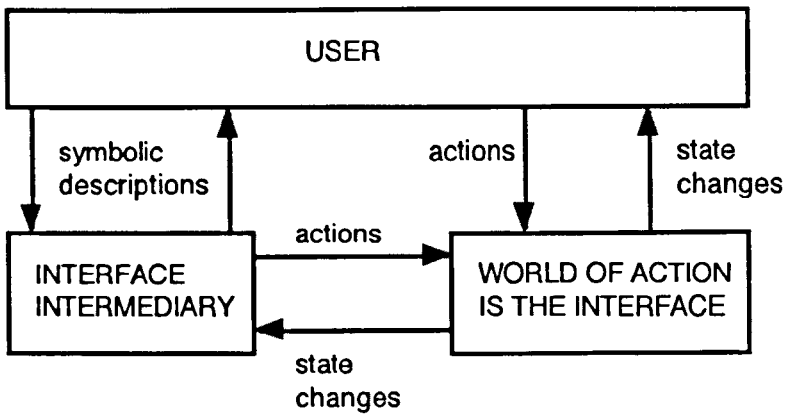


Figure 10.1 The collaborative manipulation metaphor (Hutchins, 1989: 25).

The notion that the collaborative manipulation metaphor represents the future of direct manipulation is one advocated by Frolich (1993). Shneiderman's (Shneiderman and Maes, 1997) antagonism towards agents, by contrast, means that he feels that the user should be the sole party in control of the system and that improving information visualisation should be the aim of the designer. Frolich (1993), building on recognised limitations of direct manipulation systems; and Maes (Shneiderman and Maes, 1997), arguing from the observation that file storage systems are no longer restricted to a small number of volumes on a local area network); both conclude that software agents and virtual partners are required. While the Medusa system designs do not exclude the possible inclusion of an agent as an application consistent with the model world, the file space model assumed by the first Medusa system better permits repetitive tasks and searching of the world-wide web to be conducted. These tasks use the same data structures that the user must understand and interact with in the local file space. The second version of the Medusa system, because it considers the grounding of metaphors in physical experience and because it is based on physical world metaphors, requires that any agent will be less consistent with the local model world. The agent's representation of the wider file space will be inconsistent with the user's, and it will be harder for the user to state their intentions so that the agent can act on them. By contrast, discussion of the local model world is easier. If the assumption that the image

schemata that the Medusa model world metaphors are grounded upon are common to the experience of the users is correct, then we possess a means for an agent and the user to discuss tasks and changes in the system's state. We also possess a basis for the same meaning of their respective utterances, irrespective of modality, to be inferred.

10.5.2 Metaphors for Future Computing Systems

In Chapter 2 a short review was undertaken of interesting computing systems of the sort described as *antisedentary beigeless computing* (ABC) by Underkoffler (1997) in which clear use of metaphor in their user interface is made. Marcus (1993) provides one analysis of the use of metaphor in ABC systems. In contrast to Frolich, Marcus is a proponent of the use of metaphors in user interfaces, and in future user interfaces. The principal type of ABC system that Marcus (1993) addresses are those termed *personal digital assistants* (PDAs). This class of ABC system is subject to the same criticisms of existing desk-bound systems that motivated the design of the Medusa systems.

Unlike Norman's (1998) *information appliances*, which are typically computing systems dedicated to a single information-based task with the ability to share this information with other information appliances, PDAs provide several application software packages that typically support office-based tasks in a single device. The PDA, therefore, must provide user interfaces to each of these applications within the capabilities provided by the PDA as a whole. Marcus' (1993) analysis provides metaphors (such as the Rolodex, the "to do" list, calendars, assistance, search, and selection) that a PDA must support. A basic PDA therefore presents the same problems in terms of its use of metaphor as those described in depth in Chapter 3. Where some PDAs differ from simply being a pocket-sized implementation of the desktop, however, is in preferentially supporting the *verbs* over the *nouns* that

describe users' tasks in the design of the user interface metaphors. The data objects which the applications manipulate are hidden or bound with the application's state rather than denoted as on-screen objects in their own right. One can see a key reason for this if the physical size of a typical PDA screen is compared with the screen real estate needed to implement a folders and files, or pile-based, data retrieval and storage mechanism. A focus of further work will be to examine how a Medusa-like system might be implemented for a PDA. Our experience of living with a PalmPilot[2] PDA for some time has demonstrated that support for classification and dynamic reclassification of events and data files, a topic addressed in detail in Chapter 9, is often in conflict in existing PDAs with support required to model the conceptual structures used to describe the model world.

10.5.3 Metaphor-based Design

Above, a number of new interface designs were presented, but although attempts were made to justify particular design decisions, very little was said about the impact, if any, of the development of Medusa on design practice. As with many other activities, analogy plays a part in design (Maclean et al., 1991), but we have, so far, not devoted much attention to how design of metaphor-based systems is, or should be, conducted. A small number of design methodologies for metaphor-based systems have been devised, and while they differ in the number of steps in the design process, many of the steps are common to the different methodologies. Marx (1994), Madsen (1994), and Carroll, Mack, and Kellog (1988) agree that design is a four-step process. Firstly, potential metaphors, from the user's point of view, are identified. Matches between these metaphors and underlying software are then identified with respect to representative task scenarios that the system must support.

---

[2] PalmPilot is a registered trademark of 3COM corporation.

Likely mismatches and their implications are also identified. Finally, design strategies to help users manage mismatches must be identified.

For Smyth et al. (1995) design of metaphor-based systems is based upon a far simpler model of metaphor understanding than those described in Chapters 3 and 8, and comprises six stages. Firstly the system functionality is defined, next potential vehicles (source domains) are generated and described. Vehicle-system (target domain) pairings are then analysed to identify mismatches, including conceptual baggage — user's assumptions arising from the metaphor that cannot be applied in the electronic domain. Implementation of the metaphor eventually chosen requires that the issues of representation, realism and consistency must be considered. The next step in Smyth's et al. design process is to examine and choose suitable evaluation techniques, finally, lessons learned while undertaking the design of a metaphor-based system are used to adjust the details of how process steps are performed in future design tasks.

Moll-Carrillo et al. (1995) adopt the same steps as Smyth's et al. design process. For Tscheligi and Väänänen-Vainio-Mattila (1998), design of metaphor-based systems consists of the following steps; firstly analysis of the task domain is undertaken, then mappings between sources and the target are generated. Visualisations of the sources in suitable graphical representations are then generated, the final step is to conduct evaluations of the mappings and their graphical representations. Tscheligi and Väänänen-Vainio-Mattila's work is interesting in that they, unlike others who have proposed metaphor-based design methodologies, have developed a design support environment to aid the development of metaphor-based interfaces. This tool, called ShareME, is limited, however, in that it addresses only the analysis problem — finding suitable metaphors for a task domain — and only organisational metaphors such as houses and libraries for navigation tasks are stored in its metaphor library. If tools are to be more useful they must address more steps in the design process, and

they must have access to a wider range of possible source domains. Marcus (1994: 42-43) suggests that:

> "What we shall see is not only the phenomenon of massive doses of ever changing news, sports, fashion, and tools delivered wirelessly 24 hours per day, but also constantly fluctuating 'artifacts' or 'vehicles' for the delivery of the content. User interfaces will become publications themselves ... As new metaphors emerge, older ones will disappear. The constant will be change. Imagine what it would be like if the Macintosh GUI were announced one week with world-wide flare and were gone in three months to be replaced by another variant. Welcome to the future."

Donald A. Norman[3] promises, or perhaps threatens, that "there will always be new metaphors" for user interfaces to information systems. The theories and analysis methods described above will give us ways of determining which of these new metaphors, irrespective of an overall design methodology devised, and the design support tools eventually employed to create them, can be understood by users. These methods also give us ways of suggesting consistent ways of interacting with these new systems.

---

[3] Personal communication at book signing of (Norman, 1998), London, 26th October 1998. This remark, however, contradicts sections of (Norman, 1998), see pages 180-181, which were discussed in Section 5.1.

References

Abowd, G. D. (1990) Agents: Communicating Interactive Processes. In D. Diaper, D. Gilmore, G. Cockton & B. Shackel (eds.) *Human-Computer Interaction - INTERACT'90*, North-Holland, Amsterdam: 143-148.

Abowd, G. D. (1991) *Formal Aspects of Human-Computer Interaction.* Doctoral thesis, Oxford University Computing Laboratory technical monograph PRG-97.

Adelson, B. (1989) Evaluating and Debugging Analogically Acquired Models. In G. Salvendy & M. J. Smith (eds.) *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, Elsevier Science Publishers, Amsterdam: 51-58.

Allwood, C. M. & Eliasson, M. (1987) Analogy and other Sources of Difficulty in Novices' very First Text-editing. *International Journal of Man-Machine Studies*, **27**: 1-22.

Andersen, P. B. (1997) *A Theory of Computer Semiotics.* Cambridge University Press, Cambridge.

Anderson, B., Smyth, M., Knott, R. P., Bergan, M., Bergan, J., and Alty, J. L. (1994) Minimising Conceptual Baggage: Making Choices about Metaphor. In: Cockton G., Draper, S. W., Weir G. R. S. (eds.) *People and Computers IX.* Cambridge University Press, Cambridge: 179-194.

Anderson, J. R. (1982) Acquisition of Cognitive Skills. *Psychological Review*, **89**(4): 369-406.

Anderson, J. R. (1983) *The Architecture of Cognition*, Harvard University Press, Cambridge, Massachusetts.

Anderson, J. R. (1993) *Rules of the Mind*, Lawrence Earlbaum Associates, Hillsdale, New Jersey.

Apple (1983) *A Guided Tour of Macintosh and MacWrite-MacPaint.* Tutorial disk and audio cassette. Apple Computer Inc., Cupertino, California.

Apple (1990) *Macintosh Reference.* Apple Computer Inc., Cupertino, California.

Ark, W., Dryer, D. C., Selker, T. & Zhai, S. (1998) Representation Matters: The Effect of 3D Objects and a Spatial Metaphor in a Graphical User Interface. In H. Johnson, L. Nigay & C. Roast (eds.) *People and Computers XIII*, Springer-Verlag, Berlin: 209-219.

Baecker, R., Small, I. & Mander, R. (1991) Bringing Icons to Life. In *Proceedings of CHI'91 - Human Factors in Computing Systems*, (New Orleans, Louisiana, 27 April - 2 May 1991), ACM, New York: 1-6.

Bailey, D., Feldman, J., Narayanan, S. & Lakoff, G. (1997) Modeling Embodied Lexical Development. In M. G. Shafto & P. Langley (eds.) *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*, (Stanford University, 7-10 August 1997), Lawrence Erlbaum Associates, Mahwah, New Jersey: 19-24.

Bannon, L. J. & Bødker, S. (1991) Beyond the Interface: Encountering Artifacts in Use. In J. M. Carroll (ed.) *Designing Interaction*, Cambridge University Press, Cambridge: 227-253.

Bannon, L., Cypher, A., Greenspan, S. & Monty, M. L. (1983) Evaluation and Analysis of Users' Activity Organization. In *Proceedings CHI'83 - Human Factors in Computing*, (Boston, Massachusetts, 12-15 December 1983), ACM, New York: 54-57.

Bardini, T. (2000) *Bootstrapping: Douglas Engelbart, Coevolution, and the Origins of Personal Computing*. Stanford University Press, Stanford, California.

Barnard, P. & May, J. (1995) Cinematography and Interface Design. In *Human-Computer Interaction - INTERACT'95*, (Lillehammer, Norway), Elsevier, Amsterdam: 26-31.

Barreau, D. & Nardi, B. A. (1995) Finding and Reminding: File Organisation from the Desktop. *SIGCHI Bulletin*, **27**(3): 39-43.

Barsalou, L. W. (1995) Deriving Categories to Achieve Goals. In A. Ram & D. B. Leake (eds.) *Goal-Driven Learning*, MIT Press, Cambridge, Massachusetts: 121-171.

Bass, L. & Coutaz, J. (1991) *Developing Software for the User Interface*, Addison-Wesley, Wokingham.

Bass, L., Kasabach, C., Martin, R., Siewiorek, D., Smailagic, A. & Stivoric, J. (1997) The Design of a Wearable Computer. In *Proceedings of CHI'97 - Human Factors in Computing Systems*, (Atlanta, Georgia, 22-27 March 1997), ACM, New York: 139-146.

Bederson, B. B., Hollan, J. D., Perlin, K., Meyer, J., Bacon, D. & Furnas, G. (1996) Pad++: A Zoomable Graphical Sketchpad for Exploring Alternate Interface Physics. *Journal of Visual Languages and Computing*, **7**: 3-31.

Bell, B., Rieman, J. & Lewis, C. (1991) Usability Testing of a Graphical Programming System: Things We Missed in a Programming Walkthrough. In *Proceedings of CHI'91 - Human Factors in Computing Systems*, (New Orleans, 27 April - 2 May 1991), ACM, New York: 7-12.

Benyon, D. & Imaz, M. (1999) Metaphors and Models: Conceptual Foundations of Representations in Interactive Systems Development. *Human-Computer Interaction*, **14**: 159-189.

Benyon, D., Davies, G., Keller, L., Preece, J. & Rogers, Y. (1990) *A Guide to Usability*, The Open University Press, Milton Keynes.

Bewley, W. L., Roberts, T. L., Schroit, D. & Verplank, W. L. (1983) Human Factors Testing in the Design of Xerox's 8010 "Star" Office Workstation. In *Proceedings of CHI'83 - Human Factors In Computing Systems*, (Boston, Massachusetts, 12-15 December 1983), ACM, New York: 72-77.

Bier, E. A., Stone, M. C., Fishkin, K., Buxton, W. A. S. and Baudel, T. (1994) A Taxonomy of See-Through Tools. In *Proceedings of CHI'94 - Human Factors in Computing Systems*, (Boston, Massachusetts, 24-28 April 1994), ACM, New York: 358-365.

Bier, E. A. & Stone, M. C. (1986) Snap-Dragging. *Computer Graphics*, **20**(4): 233-240.

Billingsley, P. A. (1988) Taking Panes: Issues in the Design of Windowing Systems. In M.Helander (ed.) *Handbook of Human-Computer Interaction*, North-Holland, Amsterdam: 413-436.

Blandford, A. & Green, T. R. G. (1997) OSM: An Ontology-based Approach to Usability Evaluation. In *Proceedings of a Workshop on Representations*, (Queen Mary and Westfield College, London, July 1997).

Bly, S. A. & Rosenberg, J. K. (1986) A Comparison of Tiled and Overlapping Windows. In *Proceedings CHI'86 - Human Factors in Computing Systems*, (Boston, Massachusetts, 13-17 April 1986), ACM, New York: 101-106.

Bobrow, D. G. (ed.) (1985) *Qualitative Reasoning about Physical Systems*. MIT Press, Cambridge, Massachusetts.

Bobrow, D. G. & Norman, D. A. (1975) Some Principles of Memory Schemata. In D. G. Bobrow & A. Collins (eds.) *Representation and Understanding*, Academic Press, London: 131-149.

Borg, K. (1990) IShell: A Visual UNIX Shell. In *Proceedings of CHI'90 - Human Factors in Computing Systems*, (Seattle, Washington, 1-5 April 1990), ACM, New York: 201-207.

Brewster, S. A., Wright, P. C. & Edwards, A. D. N. (1993) An Evaluation of Earcons for Use in Auditory Human-Computer Interfaces. In *Proceedings of INTERCHI'93 - Human Factors in Computing Systems*, (Amsterdam, The Netherlands, 24-29 April 1993), ACM, New York.

Brock, J. F. (1996) Whose Metaphor? *Interactions*, **3**(4): 25-35.

Brøndmo, H. P & Davenport, G. (1989) Creating and Viewing the Elastic Charles: A Hypermedia Journal. MIT Media Laboratory technical report, also in *Proceedings of Hypertext 2*, (York, 29th June 1989).

Brooks, R. (1983) Towards a Theory of the Comprehension of Computer Programs. *International Journal of Man-Machine Studies*, **18**: 543-554.

Burstein, M. H. (1986) Concept Formation by Incremental Analogical Reasoning and Debugging. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (eds.) *Machine Learning: An Artificial Intelligence Approach Volume 2*, Morgan Kaufmann, Los Altos: 351-369.

Buxton, W. A. S. (1990) A Three-State Model of Graphical Input. In D. Diaper, D. Gilmore, G. Cockton & B. Shackel (eds.) *Human-Computer Interaction - INTERACT'90*, North-Holland, Amsterdam: 449-456.

Buxton, W. A. S. (1993) *Input and Interaction: The Pragmatics of Haptic Input.* Pre-conference tutorial notes, (The National Gallery, London, 9th February 1993).

Buxton, W., Bly, S. A., Frysinger, S. P., Lunney, D., Mezrich, J. J. & Morrison, R. C. (1985) Communicating with Sound. In *Proceedings of CHI'85 - Human Factors in Computing Systems*, (San Francisco, California, 14-18 April 1985), ACM, New York: 115-119.

Byrne, M. D. (1993) Using Icons to Find Documents: Simplicity is Critical. In Proceedings of *INTERCHI'93 - Human Factors in Computing Systems*, (Amsterdam, The Netherlands, 24-29 April 1993), ACM, New York: 446-453.

Carbonell, J. G. (1983) Learning by Analogy: Formulating and Generalizing Plans from Past Experience. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (eds.) *Machine Learning: An Artificial Intelligence Approach*, Tioga, Palo Alto, California: 137-161.

Card, S. K. & Henderson, A. (1987a) A Multiple, Virtual-Workspace Interface to Support User Task Switching. In *Proceedings of CHI & GI 1987 - Human Factors in Computing Systems and Graphics Interface*, (Toronto, Canada, 5-7 April 1987), ACM, New York: 53-59.

Card, S. K. & Henderson, D. A. (1987b) Catalogues: A Metaphor for Computer Application Delivery. In H.-J. Bullinger, B. Shackel & K. Korwachs (eds.) *Human-Computer Interaction - INTERACT'87*, North-Holland, Amsterdam: 959-964.

Card, S. K., Moran, T. P. & Newell, A. (1983) *The Psychology of Human-Computer Interaction*, Lawrence Earlbaum, Hillsdale, New Jersey.

Card, S. K., Pavel, M. & Farrell, J. E. (1985) Window-based Computer Dialogues. In B. Shackel (ed.) *Human-Computer Interaction - INTERACT'84*, North-Holland, Amsterdam: 239-243.

Carpendale, M. S. T., Cowperthwaite, D. J. & Fracchia, F. D. (1995) 3-Dimensional Pliable Surfaces: For the Effective Presentation of Visual Information. In *Proceedings of the ACM Symposium on User Interface Software and Technology - UIST'95*, (Pittsburgh, Pennsylvania, 14-17 November 1995), ACM, New York: 217-226.

Carroll, J. M. (1990) *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*, MIT Press, Cambridge, Massachusetts.

Carroll, J. M. & Mack, R. L. (1985) Metaphor, Computing Systems, and Active Learning. *International Journal of Man-Machine Studies*, **22**: 39-57.

Carroll, J. M. & Mazur, S. A. (1986) LisaLearning. *IEEE Computer*, **19**(11): 35-49.

Carroll, J. M. & Olson, J. R. (1988) Mental Models in Human-Computer Interaction. In M. Helander (ed.) *Handbook of Human-Computer Interaction*, Elsevier/North-Holland, Amsterdam: 45-65.

Carroll, J. M. & Rosson, M. B. (1987) Paradox of the Active User. In J. M. Carroll (ed.) *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, MIT Press, Cambridge, Massachusetts: 80-111.

Carroll, J. M. & Thomas, J. C. (1982) Metaphor and the Cognitive Representation of Computing Systems. *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-12**(2): 107-116.

Carroll, J. M., Mack, R. L. & Kellogg, W. A. (1988) Interface Metaphors and User Interface Design. In M. Helander (ed.) *Handbook of Human-Computer Interaction*, North-Holland, Amsterdam.

Chee, Y. S. (1993) Applying Gentner's Theory of Analogy to the Teaching of Computer Programming. *International Journal of Man-Machine Studies*, **38**: 347-368.

Clark, D. M. S. (1993) *Self-Explanatory Objects: An Investigation of Object-Based Help*. Unpublished doctoral thesis, Institute of Educational Technology, The Open University, Milton Keynes.

Cockton, G. (1992) *Architecture and Abstraction in Interactive Systems*. PhD Thesis, Department of Computing Science, Heriot-Watt University, Edinburgh.

Coutaz, J. (1987) PAC, an Object Oriented Model for Dialog Design. In Bullinger, H-J. and Shackel, B. (eds.) *Human-Computer Interaction - INTERACT'87*, North-Holland, Amsterdam.

Cypher, A. (1990) Managing the Mundane. In B. Laurel (ed.) *The Art of Human-Computer Interface Design*, Addison-Welsey, Wokingham.

Czerwinski, M. P., van Dantzich, M., Robertson, G. & Hoffman, H. (1999) The Contribution of Thumbnail Image, Mouse-over Text and Spatial Location Memory to Web Page Retrieval in 3D. In M. A. Sasse & C. Johnson (eds.) *Human-Computer Interaction - INTERACT'99*, IOS Press, London: 163-170.

Davis, E. (1996) Osmose. *Wired UK edition*, **2.08**, August 1996.

de Kleer, J. & Brown, J. S. (1984) A Qualitative Physics based on Confluences. *Artificial Intelligence*, **24**: 7-84.

Decortis, F., de Keyser, V., Cacciabue, P. C. & Volta, G. (1991) The Temporal Dimension of Man-Machine Interaction. In G. R. S. Weir & J. L. Alty (eds.) *HCI and Complex Systems*, Academic Press, London: 51-72.

Demasco, P., Newell, A. F. & Arnott, J. L. (1994) The Application of Spatialization and Spatial Metaphor to Augmentive and Alternative Communication. In *Proceedings of the ACM Conference on Assistive Technology - ASSETS'94*, ACM, New York: 31-38.

Dennett, D. C. (1978) Intentional Systems. In *Brainstorms: Philosophical Essays on Mind and Psychology*, Penguin, London: 1-22.

Desurvire, H. W., Kondziela, J. M. & Atwood, M. E. (1992) What is Gained and Lost when using Evaluation Methods other than Empirical Testing. In A. Monk, D. Diaper & M. D. Harrison (eds.) *People and Computers VII*, Cambridge University Press: 89-102.

Dix, A. J. (1987) The Myth of the Infinitely Fast Machine. In D. Diaper & R. Winder (eds.) *People and Computers III*, Cambridge University Press, Cambridge: 215-228.

Dix, A. J. (1991) *Formal Methods for Interactive Systems*, Academic Press, London.

Dix, A. J. & Abowd, G. D. (1995) Delays and Temporal Incoherence due to Mediated Status-Status Mappings. In C. Johnson & P. Gray (eds.) *Papers from a Workshop on Temporal Aspects of Usability*, (University of Glasgow, 6 July 1995), Department of Computing Science, University of Glasgow, tecnical report GIST-G95-1.

Dix, A. J., Rodden, T. & Sommerville, I. (1996) A Modal Model of Versions. In *Proceedings of a BCS-FACS Workshop on Formal Aspects of the Human-Computer Interface*, (Sheffield, 10-12 September 1996): 100-109.

Douglas, S. A. & Moran, T. P. (1983) Learning Test Editor Semantics by Analogy. In *Proceedings of CHI'83 - Human Factors in Computing Systems Proceedings*, (Boston, Massachusetts, 12-15 December 1983), ACM, New York: 207-211.

Dourish, P. (1995) Developing a Reflective Model of Collaborative Systems. *ACM Transactions on Computer-Human Interaction*, **2**(1): 40-63.

Dourish, P. (2001) *Where the Action Is: The Foundations of Embodied Interaction.* The MIT Press, Cambridge, Massachusetts.

Dourish, P. & Button, G. (1998) On "Technomethodology": Foundational Relationships Between Ethnomethodology and System Design. *Human-Computer Interaction*, **13**: 395-432.

Du Boulay, J. B. H., O'Shea, T. & Monk, J. (1981) The Black Box inside the Glass Box: Presenting Computing Concepts to Novices. *International Journal of Man-Machine Studies*, **14**: 237-249.

Evanson, S. & Holland, S. (1996) Script of narration to "The Front Desk", British Broadcasting Corporation television programme to accompany the Open University course "M206 - Computing: An Object-Oriented Approach".

Fernström, M. & Bannon, L. (1997) Explorations in Sonic Browsing. In H. Thimbleby, B. O'Conaill & P. Thomas (eds.) *People and Computers XII*, Springer-Verlag, Berlin: 117-132.

Fertig, S., Freeman, E. & Gelernter, D. (1996) "Finding and Reminding" Reconsidered. *SIGCHI Bulletin*, **28**(1): 66-69.

Fisher, S. S., McGreevy, M., Humphries, J. & Robinett, W. (1986) Virtual Environment Display System. In *Proceedings of an ACM Workshop on Interactive 3D Graphics*, (Chapel Hill, North Carolina, 23-24 October 1986), ACM, New York.

Fitzmaurice, G. W., Ishii, H. & Buxton, W. (1995) Bricks: Laying the Foundations for Graspable User Interfaces. In *Proceedings of CHI'95 - Human Factors in Computing Systems*, (Denver, Colorado, 7-11 May 1995), ACM, New York:

Forbus, K. D. (1984) Qualitative Process Theory. *Artificial Intelligence*, **24**: 85-116.

Forbus, K. D. (2001) Exploring Analogy in the Large. In D. Gentner, K. J. Holyoak & B. N. Kokinov (eds.) *The Analogical Mind: Perspectives from Cognitive Science.* MIT Press, Cambridge, Massachusetts: 23-58.

Forbus, K. D. & Gentner, D. (1986) Learning Physical Domains: Toward a Theoretical Framework. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (eds.) *Machine Learning: An Artificial Approach Volume 2*, Morgan Kaufmann, Los Altos, California: 311-348.

Forbus, K. D., Gentner, D. & Law, K. (1994) MAC/FAC: A Model of Similarity-based Retrieval. *Cognitive Science*, **19**: 141-205.

Foss, D. J., Rosson, M. B. & Smith, P. L. (1982) Reducing Manual Labor: An Experimental Analysis of Learning Aids for a Text Editor. In *Proceedings of the Conference on Human Factors in Computing Systems*, (Gaithersburg, Maryland, 15-17 March 1982), ACM, New York: 332-336.

Fox, C. & Gonzalez, V. (1989) The Notebook: A New Model for the User Interface. In G. Salvendy & M. J. Smith (eds.) *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, Elsevier, Amsterdam: 620-626.

Fraser, B. (1993) The Interpretation of Novel Metaphors. In A. Ortony (ed.) *Metaphor and Thought 2nd edition*, Cambridge University Press, Cambridge: 329-341

French, R. M. (1995) *The Subtlety of Sameness: A Theory and Computer Model of Analogy-Making*, MIT Press, Cambridge, Massachusetts.

Frolich, D. M. (1993) The History and Future of Direct Manipulation. *Behaviour and Information Technology*, **12**(6): 315-329.

Furnas, G. W., Landauer, T. K., Gomez, L. M. & Dumais, S. T. (1987) The Vocabulary Problem in Human-System Communication. *Communications of the ACM*, **30**(11): 964-971.

Galloway, J. P. (1993) Helping Teachers Learn Computing with Analogies: Weak or Strong. In N. Estes & M. Thomas (eds.) *Proceedings of the 10th International Conference on Technology and Education*, (Cambridge, Massachusetts, 21-24 March 1993).

Gaver, W. W. (1986) Auditory Icons: Using Sound in Computer Interfaces. *Human-Computer Interaction*, **2**: 167-177

Gaver, W. W. (1989) The SonicFinder: An Interface that uses Auditory Icons. *Human-Computer Interaction*, **4**: 67-94

Gaver, W. W. (1991) Sound Support for Collaboration. In *Proceedings of ECSCW'91*, (Amsterdam, 25-27 September 1991).

Gaver, W. W. & Smith, R. B. (1990) Auditory Icons in Large-Scale Collaborative Environments. In D. Diaper, D. Gilmore, G. Cockton & B. Shackel (eds.) *Human-Computer Interaction - INTERACT'90*, North-Holland, Amsterdam: 735-740.

Gaver, W. W., Smith, R. B. & O'Shea, T. (1991) Effective Sounds in Complex Systems: The ARKola Simulation. In *Proceedings of CHI'91 - Human Factors in Computing Systems*, (New Orleans, Louisiana, 28 April - 2 May 1991), ACM, New York.

Gentner, D. (1983) Structure-Mapping: A Theoretical Framework for Analogy. *Cognitive Science*, **7**:155-170.

Gentner, D., Bowdle, B. F., Wolff, P. & Boronat, C. (2001) Metaphor is like Analogy. In D. Gentner, K. J. Holyoak & B. N. Kokinov (eds.) *The Analogical Mind: Perspectives from Cognitive Science*, MIT Press, Cambridge, Massachusetts: 199-254.

Gentner, D. & M. Jeziorski, M. (1993) The Shift from Metaphor to Analogy in Western Science. In A. Ortony (ed.) *Metaphor and Thought 2nd Edition*, Cambridge University Press, Cambridge: 447-480.

Gentner, D. & Stevens, A. L. (eds.) (1983) *Mental Models*, Lawrence Earlbaum Associates, Hillsdale, New Jersey.

Gibbs, R. W. (1993) Process and Products in Making Sense of Tropes. In A. Ortony (ed.) *Metaphor and Thought 2nd Edition*, Cambridge University Press, Cambridge: 252-276.

Gick, M. L. & Holyoak, K. J. (1980) Analogical Problem Solving. *Cognitive Psychology*, **12**: 306-355.

Gick, M. L. & Holyoak, K. J. (1983) Schema Induction and Analogical Transfer. *Cognitive Psychology*, **15**: 1-38.

Gifford, D. K., Jouvelot, P., Sheldon, M. & OToole, J. (1991) Semantic File Systems. In *Proceedings of the 13th ACM Symposium on Operating System Principles*.

Gittins, D. (1986) Icon-based Human-Computer Interaction. *International Journal of Man-Machine Studies*, **24**: 519-543.

Grossberg, M., Wiesen, R. A. & Yntema, D. B. (1976) An Experiment on Problem Solving with Delayed Computer Response. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-6**(3): 219-222.

Halasz, F. & Moran, T. P. (1982) Analogy Considered Harmful. In *Proceedings of the Conference on Human Factors in Computer Systems*, (Gaithersburg, Maryland, 15-17 March 1982), ACM, New York: 383-386.

Halasz, F. G. & Moran, T. P. (1983) Mental Models and Problem Solving in Using a Calculator. In *Proceedings of CHI'83 - Human Factors in Computing Systems*, (Boston, Massachusetts, 12-15 December 1983), ACM, New York: 212-216.

Halasz, F. G., Moran, T. P. & Trigg, R. H. (1987) NoteCards in a Nutshell. In *Proceedings of CHI & GI 1987 - Human Factors in Computing Systems and Graphics Interface*, (Toronto, Canada, 5-7 April 1987), ACM, New York.

Halfhill, T. R. (1997) Good-bye, GUI: Hello, NUI. *Byte*, **22**(7): 60-72.

Hall, R. P. (1989) Computational Approaches to Analogical Reasoning: A Comparative Analysis. *Artificial Intelligence*, **39**: 39-120.

Harrison, M., Roast, C. & Wright, P. (1989) Complementary Methods for the Iterative Design of Interactive Systems. In G. Salvendy & M. Smith (eds.) *Designing and Using Human-Computer Interfaces and Knowledge-based Systems*, Elsevier, Amsterdam: 651-658.

Hayes, P. J. (1985) Naive Physics I: Ontology for Liquids. In J. Hobbs & B. Moore (eds.) *Formal Theories of the Commonsense World*, Ablex, Norwood, New Jersey: 71-108.

Heckel, P. (1996) Debunking the Software Patent Myths. In P. Ludlow (ed.) *High Noon on the Electronic Frontier*. MIT Press, Cambridge, Massachusetts: 63-108.

Henderson, D. A. & Card, S. K. (1986) Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface. *ACM Transactions on Graphics*, **5**(3): 211-243.

Henry, T. R. & Hudson, S. E. (1990) Multidimenional Icons. *ACM Transactions on Graphics*, **9**(1): 133-137.

Hill, R. D. (1992) Languages for the Construction of Multi-User Multi-Media Synchronous (MUMMS) Applications. In B. A. Myers (ed.) *Languages for Developing User Interfaces*. Jones and Bartlett, London.

Hix, D. & Hartson, H. R. (1993) *Developing User Interfaces*. John Wiley and Sons, Chichester.

Hofstadter, D. R. (1985) Analogies and Roles in Human and Machine Thinking. In *Metamagical Themas*, Basic Books, New York: 547-604.

Holland, J. H., Holyoak, K. J., Nisbett, R. E. & Thagard, P. R. (1986) *Induction: Processes of Inference, Learning, and Discovery*. MIT Press, Cambridge, Massachusetts.

Holyoak, K. J. & Thagard, P. (1989) Analogical Mapping by Constraint Satisfaction. *Cognitive Science*, **13**: 295-355.

Houde, S. & Salomon, G. (1993) Working Towards Rich & Flexible File Representations. In *Adjunct Proceedings of INTERCHI'93 - Human Factors in Computing Systems*, (24-29 April 1993, Amsterdam, The Netherlands), ACM, New York: 9-10.

Howard, S. & Murray, D. M. (1987) A Taxonomy of Evaluation Techniques for HCI. In H.J. Bullinger & B. Shackel (eds.) *Human-Computer Interaction — INTERACT'87*, Elsevier Science B.V./North-Holland, Amsterdam: 453-459.

Howes, A. and Young, R. M. (1991) Predicting the Learnability of Task-Action Mappings. In *Proceedings of CHI'91 - Human Factors in Computing Systems*, (New Orleans, Louisiana, 27 April - 2 May 1991), ACM, New York: 113-118.

Hutchins, E. (1989) Metaphors for Interface Design. In M. M. Taylor, F. Nèel & D. G. Bouwhuis (eds.) *The Structure of Multimodal Dialogue*, Elsevier/North-Holland, Amsterdam: 11-28.

Hutchins, E. L., Hollan, J. D. & Norman, D. A. (1986) Direct Manipulation Interfaces. In D. A. Norman & S. W. Draper (eds.) *User Centered System Design*, Lawrence Earlbaum Associates, Hillsdale, New Jersey: 87-124.

Indurkhya, B. (1992) *Metaphor and Cognition*, Kluwer Academic, Dordrecht, The Netherlands.

Ishii, H. & Ullmer, B. (1997) Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In *Proceedings of CHI'97 - Human Factors in Computing Systems*, (Atlanta, Georgia, 22-27 March 1997), ACM, New York: 234-241.

Jeannerod, M. (1997) *The Cognitive Neuroscience of Action*, Blackwell, Oxford.

Jeffries, R., Miller, J. R., Wharton, C. & Uyeda, K. M. (1991) User Interface Evalation in the Real World: A Comparison of Four Techniques. In *Proceedings of CHI'91 - Human Factors in Computing Systems*, (New Orleans, 27 April - 2 May 1991), ACM, New York: 119-124.

John, B. E. & Packer, H. (1995) Learning and Using the Cognitive Walkthrough Method: A Case Study Approach. In *Proceedings of CHI'95 - Human Factors in Computing Systems*, (Denver, Colorado, 7-11 May 1995), ACM, New York: 429-436.

Johnson, C. & Gray, P. [eds.] (1995) *Papers from a Workshop on Temporal Aspects of Usability*, (University of Glasgow, 6 July 1995), Department of Computing Science, University of Glasgow, technical report GIST-G95-1.

Johnson, G. J. (1994) Of Metaphor and the Difficulty of Computer Discourse. *Communications of the ACM*, **37**(12): 97-102.

Johnson, J., Roberts, T. L., Verplank, W., Smith, D. C., Irby, C. H., Beard, M. & Mackey, K. (1989) The Xerox Star: A Retrospective. *IEEE Computer*, **22**(9): 11-28.

Johnson, M. (1987) *The Body in the Mind: The Bodily Basis of Meaning, Imagination and Reason*, University of Chicago Press, Chicago.

Jones, W. P. & Dumais, S. T. (1986) The Spatial Metaphor for User Interfaces: Experimental Tests of Reference by Location versus Name. *ACM Transactions on Office Information Systems*, **4**(1): 42-63.

Kaelbling, L. P. (1993) *Learning in Embedded Systems*, MIT Press, Cambridge, Massachusetts.

Kandogan, E. & Shneiderman, B. (1997) Elastic Windows: Evaluation of Multi-window Operations. In *Proceedings of CHI'97 - Human Factors in Computing Systems*, (Atlanta, Georgia, 22-27 March 1997), ACM, New York: 250-257.

Karat, C.-M., Campbell, R. & Fiegel, T. (1992) Comparison of Empirical Testing and Walkthrough Methods in User Interface Evaluation. In *Proceedings of CHI'92 — Human Factors in Computing Systems*, (Monterey, California, 3-7 May 1992), ACM, New York: 397-404.

Katz, S. D. (1991) *Film Directing Shot by Shot: Visualizing from Concept to Screen.* Michael Weise Productions, Studio City.

Kay, A. C. (1969) *The Reactive Engine*. Doctoral thesis, University of Utah, Salt Lake City, Utah, USA.

Kay, A. C. (1993) The Early History of Smalltalk. *ACM SIGPLAN Notices*, **28**(3): 69-95.

Keane, M. T., Ledgeway, T. & Duff, S. (1994) Constraints on Analogical Mapping: A Comparison of Three Models. *Cognitive Science*, **18**: 387-438.

Keiras, D. (1992) Diagrammatic Displays for Engineering Systems: Effects on Human Performance in Interacting with Malfunctioning Systems. *International Journal of Man-Machine Systems*, **36**: 861-895.

Keiras, D. E. & Bovair, S. (1984) The Role of a Mental Model in Learning to Operate a Device. *Cognitive Science*, **8**: 255-273.

Keiras, D. E. & Polson, P. G. (1983) A Generalized Transition Network Representation for Interactive Systems. In *Proceedings of CHI'83 - Human Factors in Computing Systems*, (Boston, Massachusetts, 12-15 December 1983), ACM, New York: 103-106.

Keiras, D. E. & Polson, P. G. (1985) An Approach to the Formal Analysis of User Complexity. *International Journal of Man-Machine Studies*, **22**: 365-394.

Kirsch, D. (1996) The Intelligent Use of Space. In P. E. Agre & S. J. Rosenschein (eds.) *Computational Theories of Interaction and Agency*, MIT Press, Cambridge, Massachusetts: 397-435.

Kobayashi, M. & Schmandt, C. (1997) Dynamic Soundscape: Mapping Time to Space for Audio Browsing. In *Proceedings of CHI'97 - Human Factors in Computing Systems*, (22-27 March 1997, Atlanta, Georgia), ACM, New York: 194-201.

Köhler, H. (1987) The Space-Concept and the Control of Space. In H.-J. Bullinger, B. Shackel & K. Kornwachs (eds.) *Human-Computer Interaction - INTERACT'87*, North-Holland, Amsterdam: 223-227.

Kosslyn, S. M. (1994) *Image and Brain: The Resolution of the Imagery Debate*, MIT Press, Cambridge, Massachusetts.

Kuipers, B. (1984) Commonsense Reasoning about Causality: Deriving Behaviour from Structure. *Artificial Intelligence*, **24**: 169-204.

Kurtenbach, G. & Buxton, W. (1991) Issues in Combining Marking and Direct Manipulation Techniques. In *Proceedings of the ACM Symposium on User Interface Software and Technology - UIST'91*, (Hilton Head, South Carolina, 11-13 November 1991), ACM, New York: 137-144.

Lakoff, G. (1987) *Women, Fire and Dangerous Things: What Categories Reveal about the Mind*, University of Chicago Press, London.

Lakoff, G. (1993) The Contemporary Theory of Metaphor. In A Ortony (ed.) *Metaphor and Thought 2nd edition*, Cambridge University Press, Cambridge: 202-251.

Lakoff, G. & Johnson, M. (1980) *Metaphors We Live By*, University of Chicago Press, London.

Lakoff, G. & Johnson, M. (1999) *Philosophy in the Flesh: The Embodied Mind and its Challenge to Western Thought*, Basic Books, New York.

Landauer, T. K. (1989) Relations between Cognitive Psychology and Computer System Design. In J. M. Carroll (ed.) *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, MIT Press, Cambridge Massachusetts: 1-25.

Landauer, T. K. (1995) *The Trouble with Computers: Usefulness, Usability and Productivity*, MIT Press, Cambridge, Massachusetts.

Lansdale, M. W. & Ormerod, T. C. (1994) *Understanding Interfaces: A Handbook of Human-Computer Dialogue*, Academic Press, London.

Laurel, B. (1993) *Computers as Theatre*, Addison-Wesley, Reading, Massachusetts.

Lee, A. Y., Foltz, P. W. & Polson, P. G. (1994) Memory for Task-Action Mappings: Mnemonics, Regularity and Consistency. *International Journal of Human-Computer Studies*, **40**: 771-794.

Levy, S. (1994) *Insanely Great*, Viking, New York.

Long, J. & Dowell, J. (1989) Conceptions of the Discipline of HCI: Craft, Applied Science, and Engineering. In A. Sutcliffe & L. Macauley (eds.) *People and Computers V*, Cambridge University Press, Cambridge: 9-32.

Lüdtke, M. & Nackunstz, I. (1987) User Interfaces to a Medical Archiving and Communication System. In Bullinger, H.-J., Shackel, B. & Korwachs, K. (eds.) *Human-Computer Interaction - INTERACT 87*, North-Holland, Amsterdam: 637-642.

Lundell, J. & Anderson, S. (1995) Designing a "Front Panel" for Unix: The Evolution of a Metaphor. In *Proceedings of CHI'95 - Human Factors in Computing Systems*, (Denver, Colorado, 7-11 May 1995), ACM, New York: 573-579.

Mac Cormac, E. R. (1985) *A Cognitive Theory of Metaphor*, MIT Press, Cambridge, Massachusetts.

Mackinlay, J., Card, S. K. & Robertson, G. G. (1990) A Semantic Analysis of the Design Space of Input Devices. *Human-Computer Interaction*, **5**: 145-190.

MacLean, A., Bellotti, V., Young, R. & Moran, T. (1991) Reaching Through Analogy: A Design Rationale Perspective on Roles of Analogy. In *Proceedings of CHI'91 - Human Factors in Computing Systems*, (New Orleans, Louisiana, 27 April - 2 May 1991), ACM, New York: 167-172.

Mackinlay, J. D., Robertson, G. G. & Card, S. K. (1991) The Perspective Wall: Detail and Context Smoothly Integrated. In *Proceedings of CHI'91 - Human Factors in Computing Systems*, (New Orleans, Louisiana, 27 April - 2 May 1991), ACM, New York: 173-179.

Malone, T. W. (1983) How do People Organize Their Desks? Implications for the Design of Office Information Systems. *ACM Transactions on Office Information Systems*, **1**(1): 99-112.

Maloney, J. H. & Smith, R. B. (1995) Directness and Liveness in the Morphic User Interface Construction Environment. In *Proceedings of the ACM Symposium on User Interface Software and Technology - UIST'95*, (14-17 November 1995, Pittsburgh, Pennsylvania), ACM, New York.

Mander, R., Salomon, G. & Wong, Y. (1992) A 'Pile' Metaphor for Supporting Casual Organisation of Information. In *Proceedings of CHI'92 - Human Factors in Computing Systems*, 3-7 May 1992, Monterey, California), ACM, New York: 627-634.

Marcus, A. (1993) Human Communication Issues in Advanced UIs. *Communications of the ACM*, **36**(4): 101-109.

Marx, A. N. (1994) Using Metaphor Effectively in User Interface Design. In *CHI'94 - Human Factors in Computing Systems Conference Companion*, (Boston, Massachusetts, 24-28 April 1994), ACM, New York: 379-380.

May, J. (1993) *Do Cognitive Walkthroughs Get Anywhere?* ESPRIT Basic Research Action 7040 - AMODEUS report UM/WP7.

Mayer, R. E. (1976) Some Conditions of Meaningful Learning for Computer Programming: Advance Organizers and Subject Control of Frame Order. *Journal of Educational Psychology*, **68**(2): 143-150.

Mayer, R. E. (1981) The Psychology of How Novices Learn Computer Programming. *ACM Computing Surveys*, **13**(1): 121-141.

Mayes, J. T., Draper, S. W., McGregor, A. M. & Oatley, K. (1988) Information Flow in a User Interface: The Effect of Experience and Context on the Recall of MacWrite Screens. In D. M. Jones & R. Winder (eds.) *People and Computer IV*, Cambridge University Press: 275-289.

Meira, S. L., Cavalcanti, A. L. C. & Santos, C. S. (1994) The Unix Filing System: A MooZ Specification. In K. Lano & H. Haughton (eds.) *Object-Oriented Specification Case Studies*. Prentice-Hall International, Wokingham: 80-109.

Mitchell, M. (1993) *Analogy Making as Perception: A Computer Model*, MIT Press, Cambridge, Massachusetts.

Mitchell, M. & Hofstadter, D. R. (1990) The Emergence of Understanding in a Computer Model of Concepts and Analogy-Making. *Physica D*, **42**: 322-334.

Mitchell, M. & Hofstadter, D. R. (1995) Perspectives on Copycat: Comparisons with Recent Work. In D. R. Hofstadter and the Fluid Analogies Research Group, *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*, Basic Books, New York.

Monk, A. F. & Dix, A. J. (1987) Refining Early Design Decisions with a Black-Box Model. In D. Diaper & R. Winder (eds.) *People and Computers III*, Cambridge University Press: 147-158.

Moran, T. P. (1981) The Command Language Grammar: A Representation for the User Interface of Interactive Computer Systems. *International Journal of Man-Machine Studies*, **15**: 3-50.

Morgan, C. & Sufrin, B. (1984) Specification of the Unix Filing System. *IEEE Transactions on Software Engineering*, **SE-10**(2): 128-142.

Muller, M. J. (1988) Multifunctional Cursor for Direct Manipulation User Interfaces. In *Proceedings of CHI'88 - Human Factors in Computing Systems*, (Washington, DC, 15-19 May 1988), ACM, New York: 89-94.

Myers, B. A. (1985) The Importance of Percent-done Progress Indicators for Computer-Human Interaction. In *Proceedings of CHI'85 - Human Factors of Computing Systems*, (San Francisco, California, 14-18 April 1985), ACM, New York: 11-17.

Myers, B. A. (1988) *Creating User Interfaces by Demonstration*, Academic Press, London.

Narayanan, S. (1997) Talking the Talk *is* Like Walking the Walk: A Computational Model of Verbal Aspect. In M. G. Shafto & P. Langley (eds.) *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*, (Stanford University, 7-10 August 1997), Lawrence Erlbaum Associates, Mahwah, New Jersey: 548-553.

Nardi, B. & Barreau, D. (1997) "Finding and Reminding" Revisited: Appropriate Metaphors for File Organization at the Desktop. *SIGCHI Bulletin*, **29**(1).

Nardi, B. A. & Zarmer, C. L. (1993) Beyond Models and Metaphors: Visual Formalisms in User Interface Design. *Journal of Visual Languages and Computing*, **4**: 5-33.

Nielsen, J. (1993) *Usability Engineering*, AP Professional/Academic Press, Boston, Massachusetts.

Norman, D. A. (1981) The Trouble with Unix: The System Design is Elegant but the User Interface is Not. *Datamation*, **27**(12): 139-150.

Norman, D. A. (1983) Some Observations on Mental Models. In D. Gentner & A. L. Stevens (eds.) *Mental Models*, Lawrence Erlbaum Associates, Hillsdale, New Jersey: 7-14.

Norman, D. A. (1984) Stages and Levels in Human-Machine Interaction. *International Journal of Man-Machine Studies*, **21**: 365-375

Norman, D. A. (1998) *The Invisible Computer*, MIT Press, Cambridge, Massachusetts.

O'Donnell, P. & Draper, S. W. (1995) How Machine Delays Change User Strategies. In G. Allen, J. Wilkinson & P. Wright (eds.) *Adjunct Proceedings of HCI'95*, (Huddersfield, UK, 29 August -1 September 1995).

Olsen, D. R. (1998) *Developing User Interfaces*, Morgan Kaufman, San Francisco, California.

Olson, J. S. (1992) The What and Why of Mental Models in Human-Computer Interaction. In *Proceedings of Mental Models and Everyday Activity, 2nd Interdisciplinary Workshop on Mental Models*, (Cambridge, 23-25 March 1992).

Ortony, A. (1979) Beyond Literal Similarity. *Psychological Review*, **86**(3): 161-180.

Owen, D. (1986) Naive Theories of Computation. In D. A. Norman & S. W. Draper (eds.) *User Centered System Design: New Perspectives on Human-Computer Interaction*, Lawrence Earlbaum Associates, Hillsdale, New Jersey.

Payne, S. J. (1988) Metaphorical Instruction and the Early Learning of an Abbreviated-Command Computer System. *Acta Psychologica*, **69**: 207-230.

Payne, S. J. (1991a) A Descriptive Study of Mental Models. *Behaviour and Information Technology*, **10**(1): 3-21.

Payne, S. J. (1991b) Display-Based Action at the User Interface. *International Journal of Man-Machine Studies*, **35**: 275-289.

Pérez-Quiñones, M. A. & Sibert, J. L. (1996) A Collaborative Model of Feedback in Human-Computer Interaction. In *Proceedings of CHI'96 - Human Factors in Computing Systems*, (Vancouver BC, Canada, 13-18 April 1996), ACM, New York: 316-323.

Philips, C. H. E. & Apperley, M. D. (1991) Direct Manipulation Interaction Tasks: A Macintosh-based Analysis. *Interacting with Computers*, **3**(1): 9-26.

Pinker, S. (1994) *The Language Instinct*, William Morrow Inc, New York.

Pólya, G. (1945) *How to Solve it: A New Aspect of Mathematical Method, 2nd Edition*, Penguin, London.

Polson, P. G. & Lewis, C. H. (1990) Theory-Based Design for Easily Learned Interfaces. *Human-Computer Interaction*, **5**: 191-220.

Polson, P. G., Lewis, C., Rieman, J. & Wharton, C. (1992) Cognitive Walkthroughs: A Method for Theory-based Evaluation of User Interfaces. *International Journal of Man-Machine Studies*, **36**: 741-773.

Putnam, H. (1981) *Reason, Truth and History*, Cambridge University Press, Cambridge.

Raskin, J. (2000) *The Humane Interface: New Directions for Designing Interactive Systems*. ACM Press/Addison Wesley, Reading, Massachusetts.

Reddy, M. J. (1993) The Conduit Metaphor: A Case of Frame Conflict in our Language about Language. In A. Ortony (ed.) *Metaphor and Thought 2nd Edition*, Cambridge University Press, Cambridge: 164-201.

Regier, T. (1996) *The Human Semantic Potential: Spatial Language and Constrained Connectionism*, MIT Press, Cambridge, Massachusetts.

Rekimoto, J. & Nagao, K. (1995) The World Through the Computer: Computer Augmented Interaction with Real World Environments. In *Proceedings of the ACM Symposium on User Interface Software and Technology - UIST'95*, (Pittsburgh, Pennsylvania, 14-17 November 1995), ACM, New York: 29-36.

Rieman, J., Lewis, C., Young, R. M. & Polson, P. G. (1994) "Why is a Raven like a Writing Desk?" Lessons in Interface Consistency and Analogical Reasoning from Two Cognitive Architectures. In *Proceedings of CHI'94 - Human Factors in Computing Systems*, (Boston, Massachusetts, 24-28 April 1994), ACM Press, New York.

Rizzo, A., Marchigiani, E. & Andreadis, A. (1997) The AVANTI Project: Prototyping and Evaluation with a Cognitive Walkthrough Based on the Norman's Model of Action. In Proceedings of Design of Interactive Systems — DIS'97 ACM, New York: 305-309.

Resnick, M., Martin, F., Sargent, R. & Silverman, B. (1996) Programmable Bricks: Toys to Think With. *IBM Systems Journal*, **35**(3&4): 443-452.

Rheingold, H. (1991) *Virtual Reality*, Secker & Warburg, London.

Roast, C. R. & Harrison, M. D. (1994) User Centred System Modelling Using the Template Model. In F. Paterno (ed.) *Proceedings of a Eurographics Workshop on the Design, Specification and Verification of Interactive Systems*, (Bocca di Magra, Italy, 8-10 June 1994): 261-273.

Robertson, G., Czerwinski, M., Larson, K., Robbins, D. C., Thiel, D. & van Dantzich, M. (1998) Data Mountain: Using Spatial Memory for Document Management. In *Proceedings of the ACM Symposium on User Interface Software and Technology - UIST'98*, ACM, New York: 153-162.

Rogers, Y. (1986) Evaluating the Meaningfulness of Icon Sets to Represent Command Operations. In M. D. Harrison & A. F. Monk (eds.) *People and Computers*, Cambridge University Press, Cambridge: 586-603.

Rogers, Y. (1992) Capturing Mental Models. In Y. Rogers, A. Rutherford & P. A. Bibby (eds.) *Models in the Mind: Theory, Perspective and Application*, Academic Press, London.

Rohrer, T. (1995) *Feeling Stuck in a GUI Web: Metaphors, Image-schemas, and Designing the Human Computer Interface*. Available from The University of Oregon WWW site URL http://darkwing.uoregon.edu/~rohrer/gui4web.htm

Rosch, E. H. (1973) Natural Categories. *Cognitive Psychology*, **4**: 328-350.

Rosch, E. (1978) Principles of Categorization. In E. Rosch & B. B. Lloyd (eds.) *Cognition and Categorization*, Lawrence Erlbaum Associates, Hillsdale, New Jersey: 27-48.

Rose, D. E. (1998) Beyond Search: The Information Access Research Group at Apple. *SIGCHI Bulletin*, **30**(2): 85-89.

Rose, D. E., Mander, R., Oren, T., Ponceléon, D. B., Salomon, G. & Wang, Y. Y. (1993) Content Awareness in a File System Interface: Implementing the 'Pile' Metaphor for Organising Interaction. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, New York: 260-269.

Rosenberg, J. K. & Moran, T. P. (1985) Generic Commands. In B. Shackel (ed.) *Human-Computer Interaction - INTERACT'84*, North Holland, Amsterdam.

Rumelhart, D. E. (1989) Towards a Microstructural Account of Human Reasoning. In S. Vosniadou & A. Ortony (eds.) *Similarity and Analogical Reasoning*, Cambridge University Press, Cambridge: 298-312.

Rumelhart, D. E. & Abrahamson, A. A. (1973) A Model for Analogical Reasoning. *Cognitive Psychology*, **5**:1-28.

Rumelhart, D. E. & Norman, D. A. (1981) Analogical Processes in Learning. In J. R. Anderson (ed.) *Cognitive Skills and their Acquisition*, Lawrence Earlbaum Associates, Hillsdale, New Jersey: 335-359.

Runciman, C. & Thimbleby, H. (1986) Equal Opportunity Interactive Systems. *International Journal of Man-Machine Studies*, **25**: 439-451.

Rutherford, A. & Wilson, J. R. (1992) Searching for the Mental Model in Human-Machine Systems. In Y. Rogers, A. Rutherford & P. A. Bibby (eds.) *Models in the Mind: Theory, Perspective and Application*, Academic Press, London.

Schiele, F. & Green, T. R. G. (1990) HCI Formalisms and Cognitive Psychology: The Case of Task-Action Grammar. In M. Harrison & H. Thimbleby (eds.) *Formal Methods in Human-Computer Interaction*, Cambridge University Press, Cambridge: 9-62.

Schmandt, C. & Mullins, A. (1995) AudioStreamer: Exploiting Simultaneity for Listening. In *Proceedings of CHI'95 - Human Factors in Computing Systems*, (Denver, Colorado, 7-11 May 1995), ACM Press, New York: 218-219.

Schön, D. A. (1983) *The Reflective Practitioner: How Professionals Think in Action*. Basic Books.

Sheldrake, R. (1994) *Seven Experiments That Could Change the World: A Do-It-Yourself Guide to Revolutionary Science*, Fourth Estate, London.

Shneiderman, B. (1982) The Future of Interactive Systems and the Emergence of Direct Manipulation. *Behaviour and Information Technology*, **1**: 237-256.

Shneiderman, B. (1983) Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, **16**: 57-69.

Shneiderman, B. (1986) Seven Plus or Minus Two Central Issues in Human-Computer Interaction. In *Proceedings of CHI'86 - Human Factors in Computing Systems*, (Boston, Massachusetts, 13-17 April 1986), ACM, New York: 343-349.

Shneiderman, B. (1989) Future Directions for Human-Computer Interaction. In G. Salvendy & M. J. Smith (eds.) *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, Elsevier/North-Holland, Amsterdam: 2-17.

Shneiderman, B. & Maes, P. (1997) Direct Manipulation versus Interface Agents. *Interactions*, **4**(6): 42-61.

Sibert, J. L., Hurley, W. D. & Bleser, T. W. (1986) An Object-Oriented User Interface Management System. *Computer Graphics*, **20**(4): 259-268.

Simon, H. A. (1981) *The Sciences of the Artificial 2nd Edition*, MIT Press, Cambridge, Massachusetts.

Simon, T. & Young, R. M. (1988) GOMS meets STRIPS: The Integration of Planning with Skilled Procedure Execution in Human-Computer Interaction. In D. M. Jones & R. Winder (eds.) *People and Computers IV*, Cambridge University Press, Cambridge: 581-594.

Singley, M. K. & Anderson, J. R. (1989) *The Transfer of Cognitive Skill*, Harvard University Press, Cambridge, Massachusetts.

Smith, B. C. (1996) *On the Origin of Objects*, MIT Press, Cambridge, Massachusetts.

Smith, D. C. (1977) *Pygmalion: A Computer Program to Model and Simulate Creative Thought*, Birkhauser Verlag, Basel.

Smith, D. C., Irby, C., Kimball, R., Verplank, B. & Harslem, E. (1982a) Designing the Star User Interface, *Byte*, **7**(4), April 1982.

Smith, D. C., Irby, C., Kimball, R. & Harslem, E. (1982b) The Star User Interface: An Overview. In *Proceedings of the AFIPS National Computer Conference*, (June 1982): 517-528.

Smith, R. B. (1986) The Alternate Reality Kit: An Animated Environment for Creating Interactive Simulations. In *Proceedings of the 1986 IEEE Computer Society Workshop on Visual Languages*, (Dallas, June 1986): 99-106.

Smith, R. B. (1987) Experiences with the Alternate Reality Kit: An example of the Tension Between Literalism and Magic. In *Proceedings of CHI & GI 1987 - Human Factors in Computing Systems and Graphics Interface*, (Toronto, Canada, 5-7 April 1987), ACM, New York: 61-67.

Smyth, M., Anderson, B., Knott, R. & Alty, J. L. (1995) Reflections on the Design of Interface Metaphors. In *Human-Computer Interaction - INTERACT'95*, Elsevier, Amsterdam: 339-345.

Spiro, R. J., Feltovich, P. J., Coulson, R. L. & Anderson, D. K. (1989) Multiple Analogies for Complex Concepts: Antidotes for Analogy-Induced Misconception in Advanced Knowledge Acquisition. In S. Vosniadou & A. Ortony (eds.) *Similarity and Analogical Reasoning*, Cambridge University Press, Cambridge: 498-531

Sowa, J. F. (1995) Top-Level Ontological Categories. *International Journal of Human-Computer Studies*, **43**: 669-685.

Stafford-Fraser, Q. & Robinson, P. (1996) Brightboard: A Video-Augmented Environment. In *Proceedings of CHI'96 - Human Factors in Computing Systems*, (Vancouver BC, Canada, 13-18 April 1996), ACM, New York: 134-141.

Stevens, W. R. (1992) *Advanced Programming in the UNIX Environment*, Addison-Wesley, Reading, Massachusetts.

Sutherland, I. E. (1963) *Sketchpad: A Man-Machine Graphical Communication System.* Doctoral Thesis, Massachusetts Institute of Technology, Cambridge, USA.

Tauber, M. J. (1988) On Mental Models and the User Interface. In G. C. van der Veer, T. R. G. Green, J.-M. Hoc & D. M. Murray (eds.) *Working with Computers: Theory Versus Outcome*, Academic Press, London: 89-119.

Teal, S. L. & Rudnicky, A. I. (1992) A Performance Model of System Delay and User Strategy Selection. In *Proceedings of CHI'92 - Human Factors in Computing Systems*, (Monterey, California, 3-7 May 1992), ACM Press, New York: 295-305.

Thagard, P., Holyoak, K. J., Nelson, G. & Gochfeld, D. (1990) Analog Retrieval by Constraint Satisfaction. *Artificial Intelligence*, **46**: 259-310.

Thimbleby, H. (1980) Dialogue determination. *International Journal of Man-Machine Studies* , **13**: 295-304

Thimbleby, H. (1990) *User Interface Design*, Addison-Wesley, Wokingham.

Tognazzini, B. (1992) *Tog on Interface*, Addison-Wesley, Reading, Massachusetts.

Took, R. K. (1990a) *Surface Interaction: Separating Direct Manipulation Interfaces from their Applications.* PhD Thesis available as Department of Computer Science, University of York technical report YCST 90/10.

Took, R. (1990b) Surface Interaction: A Paradigm and Model for Separating Application and Interface. In *Proceedings of CHI'90 - Human Factors in Computing Systems*, (Seattle, Washington, 1-5 April 1990), ACM, New York: 35-42.

Treglown, M. & O'Shea, T. (1993) The Computer-Computer Metaphor for Multimedia Systems. In N. Estes & M. Thomas (eds.) *Proceedings of the 10th International Conference on Technology and Education*, (Cambridge, Massachusetts, 21-24 March 1993).

Treglown, M. (1994) Qualitative Models of User Interfaces. In G. Cockton, S. W. Draper & G. R. S. Weir (eds.) *People and Computers IX*, Cambridge University Press, Cambridge.

Treglown, M. (1998) From Agents to a Networked Display Manager. In J. May, J. Siddiqi & J. Wilkinson (eds.) *HCI'98 Conference Companion - Adjunct Proceedings of the 13th British Computer Society Conference on Human Computer Interaction*, (Sheffield, 1-4 September 1998): 114-115.

Treglown, M. (1999) Is the Trashcan Being Ironic? Analysing Direct Manipulation User Interfaces using a Contemporary Theory of Metaphor. In R. Paton & I. Neilson (eds.) *Visual Representations and Interpretations*, Springer-Verlag, Berlin: 173-180.

Treglown, M. (2000) Embodiment and Interface Metaphors: Comparing Computer Filing Systems. In S. McDonald, Y. Waern & G. Cockton (eds.) *People and Computers XIV — Usability or Else!*, Springer Verlag, London: 341-356.

Treglown, M. (2001) Filing, Piling, Grabbing, and Trashing: Applying a Contemporary Theory of Metaphor to User Interface Design. In D. Rachovides & Z. Swiderski (eds,) *Proceedings of a PC-HCI'2001 workshop on Integrating Metaphors, Multimodality and Multimedia,* (University of Patras, Greece, 7 December 2001).

Tscheligi, M. & Väänänen-Vainio-Mattila, K. (1998) Metaphors in User Interface Development: Methods and Requirements for Effective Support. In D. Benyon & P. Palanque (eds.) *Critical Issues in User Interface Systems Engineering*, Springer-Verlag, London: 249-263.

Tullis, T. S. (1985) Designing a Menu-based Interface to an Operating System. In *Proceedings of CHI'85 - Human Factors in Computing Systems*, (San Francisco, California, 14-18 April 1985), ACM, New York: 79-84.

Tversky, A. (1977) Features of Similarity. *Psychological Review*, **84**(4): 327-352.

Ullmer, B. & Ishii, H. (1997) The metaDESK: Models and Prototypes for Tangible User Interfaces. In *Proceedings of the ACM Symposium on User Interface Software and Technology - UIST'97*, (Banff, Alberta, Canada), ACM, New York: 223-232.

Underkoffler, J. (1997) Antisedentary Beigeless Computing. *Personal Technologies*, **1**(1): 28-40.

Vahalia, U. (1996) *UNIX Internals*, Prentice-Hall, New Jersey.

Van Dam, A. (1997) Post-WIMP User Interfaces. *Communications of the ACM*, **40**(2): 63-67.

Vervaeke, J. & Green, C. D. (1997) Women, Fire, and Dangerous Theories: A Critique of Lakoff's Theory of Categorization. *Metaphor and Symbol*, **12**: 59-80.

Waern, Y. (1985) Learning Computerized Tasks as Related to Prior Task Knowledge. *International Journal of Man-Machine Studies*, **22**: 441-455.

Waern, Y. (1990) Human Learning of Human-Computer Interaction: An Introduction. In P. Falzon (ed.) *Cognitive Ergonomics: Understanding, Learning and Designing Human-Computer Interaction*, Academic Press, London.

Weimer, D. M. & Ganapathy, S. K. (1989) A Synthetic Visual Environment with Hand Gesturing and Voice Input. In *Proceedings of CHI'89 - Human Factors in Computing Systems*, (Austin, Texas, 30 April - 4 May 1989), ACM, New York.

Wellner, P. (1991) The DigitalDesk Calculator: Tangible Manipulation on a Desk Top Display. In *Proceedings of the ACM Symposium on User Interface Software and Technology - UIST'91,* (Hilton Head, South Carolina, 11-13 November 1991), ACM, New York: 27-33.

Wertheim, M. (1999) Out of This World. *New Scientist,* **161**(2172): 38-42.

Winograd, T. & Flores, F. (1986) *Understanding Computers and Cognition: A New Foundation for Design,* Addison-Wesley, Reading, Massachusetts.

Winston, P. H. (1980) Learning and Reasoning by Analogy. *Communications of the ACM,* **23**(12): 689-703.

Wright, P., Merriam, N. & Fields, B. (1998) From Formal Models to Empirical Evaluation and Back Again. In P. Palanque & F. Paternò (eds.) *Formal Methods in Human-Computer Interaction,* Springer-Verlag, London: 283-315.

Young, R. M. (1981) The Machine inside the Machine: User's Models of Pocket Calculators. *International Journal of Man-Machine Studies,* **15**: 51-85

Young, R. M. (1983) Surrogates and Mappings: Two Kinds of Conceptual Models for Interactive Devices. In D. Gentner & A. L. Stevens (eds.) *Mental Models,* Lawrence Earlbaum Associates, Hillsdale, New Jersey: 35-52.

Young, R. M. & Simon, T. (1987) Planning in the Context of Human-Computer Interaction. In D. Diaper & R. Winder (eds.) *People and Computers II,* Cambridge University Press, Cambridge: 363-370.

Appendix A

Qualitative Process Theory Notation and Models of Generic Processes

"One day, I had a saucepan full of water on the gas ring. Just as it was about to boil, I was suddenly called away. When I came back, 20 minutes later, the saucepan was quite empty. Now I had locked the door; the window was closed, and the room was empty except for the cat. So obviously it was the cat who drank the water."

— Erik Satie (1866-1925), from sleeve notes, *Piano Music*, EMI Records.

This appendix summarises the Qualitative Process Theory notation due to Forbus (1984) employed in this thesis. Possible QPT models of the *generic commands* that may be applied to files (Rosenberg & Moran, 1985) are also presented.

A.1 Qualitative Process Theory Notation Employed in the Thesis

`Quantity-type`	Declaration of an object attribute.
`Has-Quantity`	Declaration of an attribute possessed by a particular object type.
`Individuals`	The objects involved in, and affected by a process.
`Preconditions`	Conditions which lie outside process definitions, usually suggesting some human intervention.
`QuantityConditions`	Conditions of values of attributes that must apply before a process becomes active.
`Relations`	Relations between values of attributes of object.
`Influences`	Values are directly influenced by other values while a process is active.
`A[...]`	The amount (value) of some attribute of an object.
A_m`[...]`	The magnitude of a value (sign is ignored).
`I+(...)`	Value is directly influenced by other values. The value increases while the process is active.
`I-(...)`	Decreasing direct influence.
`(T ...)`	The proposition, or condition, is TRUE.
$\propto_{Q+}$	A value is qualitatively proportional to another. The value increases while the process is active, but the relationship is not as well defined as with a direct influence.
$\propto_{Q-}$	A value is inversely proportional to another.

A.2 QPT Models of Generic Commands

A.2.1 Moving a file

Process move-file

Individuals:
source-file an object, Has-Quantity(source-file, size)
dest-file an object
source-dir a directory
dest-dir a directory,
Has-Quantity(dest-dir, free-space)
path a data-path,
Connection(data-path,source-dir, dest-dir)

Preconditions:
(T task-is-move-file)
Aligned(path)

QuantityConditions:
A[free-space(dest-folder)] > A[size(source-file)]
A[size(source-file)] > ZERO

Relations:
Let move-rate be a quantity
A[move-rate] > ZERO
move-rate $\propto_{Q+}$ device-speed(dest-folder)
move-rate $\propto_{Q-}$ system-load

Influences:
I- (size(source-file), A[move-rate])
I+ (size(dest-file), A[move-rate])

A.2.2 Copying (or duplicating) a file

Process copy-file

Individuals:
source-file an object, Has-Quantity(source-file, size)
duplicate-file an object
source-dir a directory
dest-dir a directory,
Has-Quantity(dest-dir, free-space)
path a data-path,
Connection(data-path,source-dir, dest-dir)

Preconditions:
(T task-is-move-file)
Aligned(path)

QuantityConditions:
A[free-space(dest-dir)] > A[size(source-file)]
A[size(source-file)] > ZERO

Relations:
Let copy-rate be a quantity
A[copy-rate] > ZERO
copy-rate $\propto_{Q+}$ device-speed(dest-folder)
copy-rate $\propto_{Q-}$ system-load

Influences:
I+ (size(duplicate-file), A[copy-rate])

A.2.3 Deleting a file

Process delete-file

Individuals:
file an object, Has-Quantity(file, size)

Preconditions:
(T task-is-delete-file)

QuantityConditions:
A[size(file)] > ZERO

Relations:
Let delete-rate be a quantity
A[delete-rate] > ZERO
delete-rate $\propto_{Q-}$ system-load
delete-rate $\propto_{Q+}$ device-speed

Influences:
I- (size(file), A[delete-rate])

A.2.4 Printing a file

```
Quantity-Type(pages-to-print)
Quantity-Type(document-type)
Quantity-Type(paper)
Quantity-Type(printer-model)
Quantity-Type(pages)

process print-file

Individuals:
   doc a document,
      Has-Quantity(doc, document-type),
      Has-Quantity(doc, pages)
   myprinter a printer,
      Has-Quantity(myprinter, pages-to-print),
      Has-Quantity(myprinter, printer-model),
      Has-Quantity(myprinter, paper)
   network a datapath, Connected(doc, myprinter,
      network)

Preconditions:
   Aligned(network)
   (T task-is-print-file)

QuantityConditions:
   A[paper(myprinter)] > ZERO
   A[pages-to-print(doc)] > ZERO

Relations:
   Let print-rate be a quantity
   A[print-rate] > ZERO
   printrate ∝Q printer-model(myprinter)
   printrate ∝Q document-type(doc)

Influences:
      I-(pages-to-print(doc), print-rate)
      I-(paper(myprinter), print-rate)
```

Appendix B

Forms used to Conduct Cognitive Walkthroughs

Chemist: *Ah, certainly. Walk this way please.*

Man: *If I could walk that way, I wouldn't need aftershave.*

— Chapman et al., *Monty Python's Flying Circus: Just the Words Vol.1*, Methuen.

B.1 Forms Completed During a Walkthrough

During the second phase of a walkthrough, the walkthrough itself, a number of questions must be answered and forms completed by the individual(s) conducting the walkthrough for each action in the action sequence prepared which, if performed, would result in the successful completion of the task.. The forms for the full version of cognitive walkthrough undertaken and reported above are taken from Polson, Lewis, Rieman and Wharton (1992) and reproduced below.

B.1.1 Section One of Phase Two of a Walkthrough

Cognitive Walkthrough For A Step

Task _____ **Action #** _____

1. Goal structure for this step

1.1 Correct goals. What are the appropriate goals for this point in the interaction? Describe as for initial goals.

1.2 Mismatch with likely goals. What percentage of users will not have these goals, based on the analysis at the end of the previous step? Check each goal in this structure against your analysis at the end of the previous step. Based on that analysis, will all users have the goal at this point, or may some users have dropped it or failed to form it? Also check the analysis at the end of the previous step to see if there are unwanted goals, not appropriate for this step, that will be formed or retained by some users. (% 0 25 50 75 100)

B.1.2 Section Two of Phase Two of a Walkthrough

2. Choosing and executing the action.

Correct action at this step: _____

2.1 Availability. Is it obvious that the correct action is a possible choice here? If not, what percentage of users might miss it? (% 0 25 50 75 100)

2.2 Label. What label or description is associated with the correct action?

2.3 Link of label to action. If there is a label or description associated with the correct action, is it obvious, and is it clearly linked with this action? If not, what percentage of users might have trouble? (% 0 25 50 75 100)

2.4 Link of label to goal. If there is a label or description associated the correct action, is it obviously connected with one of the current goals for this step? How? If not, what percentage of users might have trouble? Assume all users have the appropriate goals listed in Section 1. (% 0 25 50 75 100)

2.5 No label. If there is no label associated with the correct action, how will users relate this action to a current goal? What percentage might have trouble doing so? (% 0 25 50 75 100)

2.6 Wrong choices. Are there other actions that might seem appropriate to some current goal? If so, what are they and what percentage of users might choose one of these? (% 0 25 50 75 100)

2.7 Time-out. If there is a time-out in the interface at this step does it allow time for the user to select the appropriate action? How many users might have trouble? (% 0 25 50 75 100)

2.8 Hard to do. Is there anything physically tricky about executing the action? If so, what percentage of users will have trouble? (% 0 25 50 75 100)

B.1.3 Section 3 of Phase Two of a Walkthrough

3. Modification of goal structure
Assume the correct action has been taken. What is the system's response?

3.1 Quit or backup. Will users see that they have made progress towards some current goal? What will indicate this too them? What percentage of users will not see progress and try to quit or backup? (% 0 25 50 75 100)

3.2 Accomplished goals. List all current goals that have been accomplished. Is it obvious from the system response that each has been accomplished? If not, indicate for each how many users will not realise it is complete.

3.3 Incomplete goals that look accomplished. Are there are any current goals that have not been accomplished, but might appear to have been based on the system response? What might indicate this? List any such goals and the percentage of users will (sic) think they have actually been accomplished.

3.4 "And-then" structures. Is there an "and-then" structure, and does one of its subgoals appear to be complete? If the subgoal is similar to the supergoal, estimate how many users may prematurely terminate the "and-then" structure.

3.5 New goals in response to prompts. Does the system response contain a prompt or cue that suggests any new goal or goals? If so, describe the goals. If the prompt is unclear, indicate the percentage of users who will not form these goals.

3.6 Other new goals. Are there any other new goals that users will form given their current goals, the state of the interface and their background knowledge? Why? If so, describe the goals, and indicate how many users will form them. NOTE that these goals may or may not be appropriate, so forming them may be bad or good.

Appendix C

Metaphors We Stack By

"Per [Bak] has an appealing visual analogy for a system at the critical state: a sand pile."

— Roger Lewin (1993) *Complexity: Life at the Edge of Chaos*, J. M. Dent.

C.1 Introduction

The second version of the Medusa system, discussed in Chapter 9 above, is designed to make use of a version of the pile metaphor to support casual organisation of the user's work and to support ad hoc categorisation to aid performance of the user's immediate tasks. In this appendix an analysis is undertaken using the Lakoff/Johnson theory to attempt to ground the metaphorical language used to describe user's work and organisation of information. This analysis also seeks to define the aspects of the second Medusa user interface that support file organisation tasks. As we have not yet undertaken a study of users' existing pile-related tasks (the analysis below is based upon the study undertaken by Malone (1983), who provides considerable data from the users that he studied. We quote considerably from Malone's (1983) paper below and use his text as the corpus to be analysed. The need to conduct a study of our own similar to Malone's is prompted by the analysis presented below, and would help to identify further issues and task scenarios that a computer-based pile system may need to support.

C.2 Users' Construction and Use of Piles

Malone classifies the construction and use of piles into two types, neat and messy. These categories reflect a user's job type and status in addition to their need for, and use of, information resources. This distinction is maintained below in collecting meaningful passages describing pile organisation from Malone's study in case a number of metaphors are found to be needed to describe piles and their use.

C.2.1 A Neat Office

In Malone's study, 'Michael' is said to have a neat office. Malone describes Michael's office and information usage saying:

> "As a purchasing agent, Michael's' work is based primarily on a set of standard forms. The arrangement of his office reflects the flow of these forms, and the description will focus on this flow. There are different piles and files in the office for different kinds of forms and for forms in various stages of processing. Michael summarised one aspect of this as follows:
>
> > The good stuff is all out on the table. The paperwork flow is always out. I don't put paperwork - other than the stuff that is in the suspense file - in a drawer. (M.P., 10/27/81)
>
> According to Michael's description, purchase requisitions enter his office in his in-basket (top of tray A) and he sorts them into two groups awaiting processing in pile B. Some requisitions can be processed immediately and put in the out-basket (bottom of tray A); others are kept in the 'hold' tray (middle of tray A) until further information can be collected (usually by telephone). Each morning,

Michael sorts the processed forms from the out-basket (bottom of tray A) into folders in tray D for distribution.

When his copy of a purchase order returns to Michael's in-basket, he files it in the suspense file (F) of open orders according to the date when the merchandise is supposed to be delivered. When forms confirming delivery ('receivers') arrive from the receiving department, they are temporarily placed in pile H and then matched with the purpose orders on file...Pile C contains purchase orders from file F that require some special action as a result of someone calling to check on them or change them.

The bookshelf contains primarily books and catalogues, loosely arranged. The bottom drawer of file F contains information on freight and commodities, arranged by subject. Information to be files here is also stacked in pile I and tray E. The desk file drawer includes more product information, administrative memos, and blank forms - again arranged by subject. Michael sometimes uses his blackboard to list important things to remember to do, and he has a bulletin board that contains some telephone numbers and address lists." (Malone, 1983: 101-102)

C.2.2. A Messy Office

Malone offers as an example of a messy office that of 'Kenneth', saying:

"As a research scientist, Kenneth has very little routine paper flow. Most of the information in his office consists of books, papers, magazines, personal notes, and computer listings. In contrast to

Michael's fairly neat and narrowly defined piles, Kenneth's office is filled with loosely stacked piles of mixed content. For example, here is how Kenneth describes the contents of piles A, B, C, D and E.

> *Kenneth*: Beside my terminal [piles A and B] are basically piles of stuff about what I need in hacking in the recent past. The deeper you go, the further back it is. Off to the right [gestures to piles C, D, and E] is stuff that I've shoved to the right when the pile beside my terminal got too high. But I've periodically pruned it so it's no longer useful; it's just a pile of junk....
>
> *Interviewer*:...But these things [gestures to piles A and B] - you know pretty well what's in these piles?
>
> *Kenneth*: Uh..there's probably one or two copies of the paper David and I have been working on, piles of notes on [two projects], and there's probably some other random things - documentation for computers...Here's [pulls document out of pile B and reads its title]. Actually I have a newer one sitting in the - I know there's a newer one sitting in the pile [looks through the pile A]..and I don't know where it is. Ah! here's a good one - the new one.

A similar lack of clear organisation prevails on the desk as well:

> *Kenneth*: The desk is sort of random. It's sort of mostly recent stuff, because I periodically do clean off my desk. For about 30 seconds it's clean. I usually separate it into piles that have to be instantly answered, should be answered in a week, or whatever has appropriate places. That pile there is mostly stuff that should be dealt with in a week..And it's been sitting for months.
>
> The desk mostly has right now sort of - I get infinite junk mail, subscribe to too many magazines. So a lot of that is magazine reading I haven't caught up on. And there's a few piles of critical

> stuff in there. I don't know..I'm sure when I find them, somebody will be mad at me for not answering their letter. I have a letter from Baker hidden someplace in here complaining about one of my papers. It's been here for a year and a half and I haven't answered it. (K. H. 10/16/81)

> The rest of the office has other piles of books and papers on the floor as well as on tables and shelves. The bookshelves include binders of computer documentation, technical reports, and back issues of journals, some of which are filed with cardboard dividers. There are two bulletin boards containing assorted items such as letters, phone messages, research notes, and a raffle ticket. The blackboard contains, among other things, remnants of several conversations and two partially redundant lists of things to do." (Malone, 1983: 103-104)

The two case studies that Malone details, which are quoted from above, differ in their use of piles. , In both cases, though, the exact roles of piles in the temporal order and time scales in which tasks must be performed (and in the success with which deadlines are met) are similar. Also similar are the language and metaphors used by Malone, 'Kenneth', and 'Michael' to describe interaction with piles. These metaphors will be examined in the following section.

C.3 A Logic of Piling

A fundamental image schema that describes much bodily experience, in the Lakoff/Johnson view, is the CONTAINER schema. Johnson (1987) observes that human beings constantly experience their bodies as containers and as things in

containers (for example, rooms). The notion of containment, as is captured in an image schema, is depicted in Figure C.1.

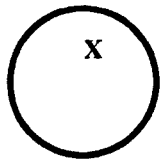


Figure C.1 A containment schema (Johnson, 1987: 23)

The container schema, in Lakoff's (1987) description has the structural elements of an interior, a boundary and an exterior, and like many image schemas its internal structure yields a basic "logic". This logic is described by Lakoff (1987: 272) as follows:

> "Everything is either inside a container or out of it — P or not P. If container A is in container B and X is in A, then X is in B — which is the basis for modus ponens: If all A's are B's and X is an A, then X is a B."

Johnson (1987: 22) identifies a number of consequences of the structure of in-out schemata of the sort that will ground understanding of actions that bring about or change instances of containment, these consequences being:

> "(i) The experience of containment typically involves protection from, or resistance to, external forces...
>
> (ii) Containment also limits and restricts forces within the container...
>
> (iii) Because of this restraint of forces, the constrained object gets a relative fixity of location...

(iv) This relative fixing of location within the container means that the contained object becomes either accessible or inaccessible to the view of some observer. It is either held so that it can be observed, or else the container itself blocks or hides the object from view.

(v) Finally, we experience transitivity of containment. If B is *in* A, then whatever is *in* B is also *in* A."

We can reveal the user's work language and the metaphors that ground understanding of a domain by applying the methods suggested by the few metaphor-based design approaches that exist. The method used by Lakoff and Johnson (Lakoff and Johnson, 1980; Lakoff, 1987; Johnson, 1987) is to catalogue actual speech production in order to highlight metaphors and to link speech to the concepts underlying it. The method is similar to that of the textual analysis used in object-oriented design, we identify metaphors by highlighting (by underlining) verbs and nouns that describe piles and interaction with them. It is possible, by examining the ways that Malone and his subjects talk about pile organisation and the gestures observed by Malone as his subjects describe their information resource organisation, to conclude that understanding of piles is based on the PILE IS CONTAINER metaphor. The first Medusa system, described in Chapter 6, examined the consequences of the FOLDER IS CONTAINER metaphor underlying document organisation in systems that implement the desktop user interface metaphor. Also in Chapter 9, the question raised by Mander, Salomon and Wong (1992) as to how their pile metaphor fitted in with the Apple implementation of the desktop metaphor as a whole was considered. As piles and folders are understood in terms of the same image schemata, one can question whether the pile is much of an advance over the traditional folder. One is able to understand users' confusion over the behaviour of piles and folders being used in the same desktop environment when files pass over, or are dropped onto, piles and folders. The image schemata that ground

understanding of both piles and folders are the same. We can suggest, therefore, that a user confronted by a folder on the top of a pile is likely to conclude that a file dropped onto the pile will be placed in the folder, the folder being the apparent target for the file when dropped. Means of overcoming this ambiguity within the second version of the Medusa system were discussed previously in Chapter 9.